programação profissional em basic



MSX
IBM-PC
MBASIC



programação profissional em basic

MSX

IBM-PC

MBASIC



Roberto Massaru Watanabe

programação profissional em basic

MSX

IBM-PC

MBASIC

1ª edição



EXPEDIENTE
Coordenação Editorial: Pierluigi Piazzi
Coordenação Didática: Betty Fromer Piazzi
Produção Editorial: Rosa Kogan Fromer
Editoração: Renato da Silva Oliveira
Digitação e Revisão: Rosa Maria X.M. Shigeno
Arte e Capa: Ana Lúcia Antico



ALEPH
Publicações e Assessoria Pedagógica Ltda.
Av.Dr.Guilherme Dumont Villares, 1523 s/4
05640 São Paulo SP - Fone: (011) 843-3202

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Watanabe, Roberto Massaru, 1946-
W294p Programação profissional em BASIC : MSX, IBM-PC, mBASIC / Roberto Massaru Watanabe. -- São Paulo : Aleph, 1988.

Bibliografia.

1. BASIC (Linguagem de programação para computadores) 2. IBM Computador pessoal - Programação 3. MSX-BASIC(Linguagem de programação para computadores) I. Título.

CDD-001.6424
-001.642

88-0930

Índices para catálago sistemático:

1. BASIC : Linguagem de programação : Computadores : Processamento de dados 001.6424
2. BASIC MSX : Linguagem de programação : Computadores : Processamento de dados 001.6424
3. IBM Computador Pessoal : Programação : Processamento de dados 001.642
4. IBM PC : Computador Pessoal : Programação : Processamento de dados 001.642

sumário

nota do editor

Informatizar uma sociedade não é, certamente, uma tarefa fácil. Informatizar a sociedade brasileira, extremamente desinformada por uma estrutura educacional catastrófica, fruto de décadas de autoritarismo e burocracia corrupta, é uma tarefa quase impossível!

Apesar disso, o nível médio da inteligência dos brasileiros não foi, felizmente, seriamente afetado pelos fatores conjunturais. Enquanto as escolas de computação, em todos os níveis, ministravam cursos no mínimo obsoletos, uma grande quantidade de brasileiros inteligentes partiram para o auto-didatismo, alfabetizando-se em pequenos e baratos microcomputadores pessoais, ensaiando seus primeiros passos de maneira talvez não metódica, mas muitas vezes brilhante, na linguagem BASIC.

Embora alguns "gênios" da computação nacional olhem esta linguagem com profundo desprezo (fruto de uma visão academicamente elitista e distorcida) uma enorme quantidade de usuários, muitos dos quais extremamente lúcidos e criativos, a utiliza por ter sido sua "porta de entrada" no mundo da informática cujo acesso, de outra forma, lhes teria sido vedado ficando restrito a uma minoria "cartorial", em muitos casos ineficiente e estéril.

Para atender às necessidades destes usuários inteligentes mas sem uma formação metódica em programação, é que convidamos o engenheiro ROBERTO MASSARU WATANABE, bem brasileiro apesar do sobrenome, para escrever este livro.

Embora a formação do Watanabe tenha sido em sistemas de grande porte, usando técnicas que transcendem a linguagem BASIC, seu enorme tirocínio em microcomputadores pessoais lhe deu a possibilidade de transferir boa parte destes recursos para esta linguagem mais simples.

Dando um passo a mais no mundo da informática o leitor terá a possibilidade de, construindo programas passo a passo com o autor, dominar as técnicas do "bem programar", tornando suas listagens inteligíveis a terceiros, permitindo uma fácil manutenção e implementação dos sistemas desenvolvidos.

O leitor deve ter consciência de que, como na famosa propaganda do "eu sou você amanhã", ele mesmo poderá não entender, daqui a alguns meses, como ele próprio desenvolveu um determinado raciocínio!

1

introdução

A cada dia que passa, mais e mais problemas passam a ser resolvidos com o emprego de computadores. Basta dar uma folheada nas revistas especializadas ou efetuar uma visita a uma livraria. O computador está transpondo os limites tradicionais das 4 paredes de um escritório e encontrando o seu lugar na medicina, na pecuária, na indústria e até em áreas pouco habituadas ao processamento de dados, como as ciências sociais e a religião.

Independentemente da natureza de um problema, dizer que o mesmo passou a ser solucionado com o auxílio de um computador, significa que houve uma feliz associação entre o computador, o problema e o usuário. Cabe lembrar que a simples reunião desses 3 componentes não consegue, por si só, encontrar a solução. É necessário termos um certo usuário que se dafronta com um determinado problema e que tenha a vontade de resolver este problema com o auxílio do computador.

Nos problemas que passaram a ser resolvidos com computadores, a quem devemos atribuir os méritos desse feito? Ao computador, cujos bits, cada vez mais poderosos, estão sendo oferecidos a preços cada vez mais acessíveis ao grande público? Ao problema que, pela sua natureza estratégica dentro da empresa, necessita de soluções rápidas mesmo que implique em custos maiores? Ao usuário que, sendo aberto às novas

tecnologias, conseguiu em pouco tempo absorver e dominar os potentes recursos oferecidos por um sistema computacional?

Analisando detalhadamente cada um dos casos em que o emprego do computador trouxe vantagens reais, constatamos que todos os componentes, isto é, o computador, o problema e o usuário têm a sua parcela de contribuição. Existe, porém, um quarto componente, geralmente esquecido, que desempenha um papel decisivo nessa história: o programa.

Sem um bom programa, os melhores computadores e os usuários mais habilitados do mundo quase nada podem fazer, nem mesmo resolver os problemas mais elementares do nosso dia a dia.

É claro que mesmo tendo uma participação pequena, o computador e o usuário são componentes que não podem ser desprezados, pois fazem parte de um mesmo contexto, e a solução com o emprego de computadores deve ser encaminhada dentro dele. Em outras palavras, não seria aceitável a afirmação de que o computador e o problema estão coerentes porém o usuário deve ser trocado.

Em qualquer situação em que desejamos empregar computadores, devemos encontrar a fórmula que permita uma participação equilibrada desses componentes e a receita que permite dosar cada uma dessas partes é o programa. Compete ao programa o reconhecimento dos dados do problema, a manipulação adequada desses dados valendo-se dos recursos oferecidos pelo computador e a apresentação da solução otimizada desse mesmo problema na visão do usuário.

É grande, pois, a responsabilidade de um programa.

A eficiência do emprego de computadores está diretamente ligada não só à potência dos algoritmos e técnicas de programação que foram utilizadas no programa, mas também à apacidade que o programa tem de amoldar-se aos hábitos e vícios dos usuários. A solução de um determinado problema pode ser apresentada de mil maneiras diferentes e a rápida compreensão dessa solução, depende da maneira em que o usuário gostaria ou está habituado a recebe tais soluções.

Por outro lado, um problema dificilmente se nos apresenta de modo estático, imutável ao longo do tempo. Existe uma espécie de evolução do problema. O programa feito para resolver esse problema deve ter a capacidade de acompanhar essa evolução.

Como se vê, existem muitos detalhes importan-

tes que devemos considerar quando vamos nos propor a desenvolver um programa.

Este livro apresenta alguns tópicos que permitirão, aos programadores em vias de profissionalização estruturar um esquema de produção de programas para que a atividade de desenvolvimento possa ser efetuada d uma maneira rápida e racionalizada, produzindo programas cuja operação possa ser executada de maneira simples por usuários leigos, sem grandes conhecimentos de programação.

Como linha central, adotou-se o BASIC da Microsoft, por ser um dos mais completos e poderosos. Existem, é claro, algumas diferenças em determinados comandos e tais diferenças estão sumarizadas no Apêndice I.

A estrutura do livro compõe-se de partes onde foram agrupadas as sugestões de mesma natureza e em cada capítulo é desenvolvido um trecho de um sistema.

No capítulo 9, os diversos trechos do sistema são agregados e o leitor terá desenvolvido o núcleo de um SISTEMA DE CONTROLE DE ESTOQUE, objetivo final do livro.

2

programas

Este capítulo analisa a aparência de uma listagem de programa, mostrando a conveniência de se distribuir as instruções e os comentários de maneira a facilitar a leitura da listagem e a interpretação da estrutura lógica do programa.

Um programa pode ser algo como o apresentado na figura 2.1.

Figura 2.1 - Programa com "estilo legível".

```
100 INPUT "Fornecer as parcelas";A,B
101 C = A + B
102 PRINT "A soma de ";A;
103 PRINT " com ";B;" = ";C
104 END
```

Como também pode ser o da figura 2.2.

Figura 2.2 - Programa com "estilo compacto".

```
473 COLOR15,1,1:SCREEN2:FORB=0TO127:X4=B
*B:M=-128:A=SQR(16384-X4):FORI=-ATOASTEP
3:R=SQR(X4+I*I)/128:F=COS(16*R)*(1-R)*2:
Y=I/5+F*32:IFY<=MTHEN20ELSEM=Y:Y=128+Y:X
=128+B:PSET(X,191-Y):X=128-B:PSET(X,191-
Y)
474 IFP=1THEN20ELSENEXTI,B:P=1:CT=15:CP=
1:GOSUB500:END
```

Independentemente do fato de o programa funcionar ou não e de ser eficiente ou não, existe um fator bastante importante que deve ser levado em consideração: caso haja necessidade de se introduzir uma modificação no programa, ela deve ser implementada no menor prazo possível.

Se o programa foi desenvolvido há pouco tempo, é bem possível que lembremos ainda da lógica que foi adotada. Neste caso, as modificações poderão ser feitas em curtíssimo prazo.

Na prática, porém, nos defrontamos com uma série de fatores que fazem com que mesmo as modificações de pequeno porte se transformem em verdadeiros pesadelos, levando, muitas vezes, a crer que seria mais fácil o desenvolvimento de um novo programa. Tais fatores podem ser os mais variados, sendo os mais usuais, os seguintes:

- O programa foi desenvolvido há muito tempo, de modo que não lembramos mais da lógica que foi adotada e nem do significado de determinadas variáveis.

- A listagem de que dispomos não corresponde exatamente ao programa em uso e que precisa ser modificado. Obter uma listagem atualizada é uma tarefa difícil pois o computador em que o programa é utilizado fica numa filial do interior e não dispõe de impressora.

- O programa é muito longo e manusear uma listagem enorme à procura de um determinado trecho que deve ser modificado se torna cansativo e demorado.

- Depois desse programa já desenvolvemos uma dezenas de outros semelhantes, de modo que confundimos com facilidade trechos de um com trechos dos outros.

- O programa foi desenvolvido por um outro programador, cujo estilo de programação é bastante diferente daquele que adotamos.

- Existem várias listagens do mesmo programa e é praticamente impossível descobrir qual é a mais atual.

Para evitar esses transtornos, procura-se confeccionar, paralelamente ao desenvolvimento do programa, um manual conhecido como Manual do Programa.

O Manual do Programa é um relatório que contém todos os dados que caracterizaram o ambiente segundo o qual foi motivado o desenvolvimento do programa. Nele são relatados os fatores que caracterizam o problema, as condições em que se fez o programa, os recursos disponíveis, o tipo de pessoal envolvido na solução do problema, o estudo das linguagens mais adequadas para o desenvolvimento do programa, as técnicas de programação e os algoritmos empregados e outras informações que possam, de alguma forma, serem úteis ou mesmo necessárias quando, no futuro, tivermos que proceder a alguma modificação.

A elaboração do Manual do Programa deve ser feita ao mesmo tempo em que se faz a elaboração do programa, para que nenhum dado seja omitido.

Na prática, porém, essa elaboração simultânea nem sempre é viável, pois durante o desenvolvimento do programa, muitas surpresas nos são reservadas pelo computador, de modo que o prazo que inicialmente estimamos para se ter o programa pronto e funcionando é consumido rapidamente e, numa época ainda distante da implantação do programa, já somos pressionados diariamente pelos usuários, ávidos pela solução via computador.

Uma das maneiras de evitar, ou ao menos diminuir, a influência desses fatores é fazer com que a listagem do programa seja fartamente comentada, valendo-se do comando REM.

Vejamos, em partes, as maneiras pelas quais os programas podem ser enriquecidos com comentários e outras informações:

IDENTIFICAÇÃO

Um programador que se dedica à confecção de programas terá, após algum tempo, um conjunto respeitável de programas. Para que serve cada um deles? Quando foi desenvolvido? Qual a última versão? Estas são apenas algumas das questões que necessitam ser respondidas quando nos lançamos na tarefa de modificar um programa.

Para facilitar a identificação de um programa usa-se agrupar as informações que o identificam no início do mesmo.

Na identificação do programa, deve-se colocar pelo menos as informações seguintes:

- nome do programa
- para que serve
- nome do autor e outros participantes
- data em que foi elaborado
- número e data da última modificação

A identificação deve ser destacada do resto da listagem. Um exemplo de uma identificação completa e padronizada é apresentada na figura 2.3.

Figura 2.3 - Documentação no próprio programa.

```
100 ' ********************************
101 ' * FOLPAG = Folha de Pagamento *
102 ' ********************************
103 '
104 '  Programa desenvolvido  para o
105 '  calculo da Folha de Pagamento
106 '
107 '     Joao da Silva - 13/10/85
108 '
109 '          R-1 28/02/86
110 '   Modificado para a adocao do
111 '          Plano Cruzado
112 '
```

A palavra REM deve ser evitada pois a sua existência na sequência de instruções de um programa, pode dificultar a visualização e a compreensão rápida de partes do programa. Veja como fica difícil a localização do comentário na listagem da figura 2.4 .

Figura 2.4 - Uso indesejável da pseudo-instução REM.

```
893 PRINT "DETERMINACAO DO MAIOR"
894 REM CALCULA X=MAIOR ENTRE 1 E 100
895 RES=AUX(1)
896 FOR I=2 TO 100
897 IF RES<AUX(I) THEN RES =XUS(I)
898 NEXT I
```

Empregando-se o apóstrofo no lugar da palavra REM e ainda inserindo-se alguns espaços, podemos fazer com que a listagem acima seja um pouco mais legível. Veja a figura 2.5.

Figura 2.5 - Uso do apóstrofo como REM.

```
893 PRINT "DETERMINACAO DO MAIOR"
894 '     CALCULA X=MAIOR ENTRE 1 E 100
895 RES=AUX(1)
896 FOR I=2 TO 100
897 IF RES<AUX(I) THEN RES =XUS(I)
898 NEXT I
```

ESTRUTURA DO PROGRAMA

A estrutura lógica do programa é o item mais importante para a perfeita e rápida compreensão do mesmo. Os diversos trechos que compõem um programa são de natureza diversas, dependendo daquilo objetivado pelo trecho.

Assim, podemos ter alguns trechos cujos objetivos sejam as principais etapas do problema e outros que procuram resolver detalhes do programa, tais como a formatação de um campo ou algoritmos introduzidos para contornar determinadas deficiências do sistema.

Um programa de Folha de Pagamento pode ser estruturado conforme o fluxograma da figura 2.6.

Como se vê, os blocos principais do programa são os seguintes:

- INÍCIO
- LEITURA DOS DADOS DE UM FUNCIONÁRIO
- CÁLCULO DOS PROVENTOS
- CÁLCULO DOS DESCONTOS
- IMPRESSÃO DOS RESULTADOS
- OUTRO FUNCIONÁRIO
- FIM

Na listagem do programa, esses blocos devem receber um destaque especial para poderem ser facilmente localizados.

O uso conveniente da tecla TAB e linhas adicionais com símbolos repetidos antes e depois do nome do parágrafo, fazem com que o nome fique em evidência no meio da listagem. Veja a figura 2.7 .

Além dos parágrafos principais, existem os parágrafos secundários que também merecem um destaque, porém com menor evidência.

Isso pode ser feito de acordo com o que foi feito para o parágrafo principal, porém empregando um símbolo de menor evidência, ou mesmo sem nenhum símbolo. Veja a figura 2.8 .

Figura 2.6 - Fluxograma do Programa Folha de Pagamento



Figura 2.7 - Uso da tabulação na documentação.

```
200 ' ------------------------------
201 ' --  CALCULO DE PROVENTOS    --
202 ' ------------------------------
```

Além desses trechos, existem, normalmente, em um programa, outros trechos que não estão relacionados diretamente com a sua lógica principal. São trechos em que são efetuadas determinadas conversões, pesquisas em tabelas, etc.

Figura 2.8 - Prioridade nas linhas REM.

```
210 '
211 '    Desconto devido ao IAPAS
212 '
```

Esses trechos devem ser programados em uma região distante do trecho principal do programa. A sua execução pode ser efetuada por um GOSUB.

Por exemplo, é sabido que o valor que corresponde ao desconto para o IAPAS é calculado por uma fórmula do tipo:

IAPAS = REFERÊNCIA * ALÍQUOTA

Nada mais correto do que incluir a fórmula no trecho principal do programa. Veja a figura 2.9 .

Figura 2.9 - Definição de fórmulas.

```
210 '
211 '    Desconto devido ao IAPAS
212 '
214 IAPAS = REFERENCIA * ALIQUOTA
```

O valor da REFERENCIA e da ALIQUOTA variam em função do salário, conforme uma tabela que é fornecida pelo IAPAS e pode sofrer alterações de um mês para outro. A estrutura principal do programa não deve ser sucetível a qualquer alteração motivada por uma causa externa à empresa. A Folha de Pagamento é um instrumento interno da empresa, e deve retratar a estrutura de organização da mesma, e se existe um desconto devido ao IAPAS, este é motivado por uma causa externa à empresa. Por isso, o critério de cálculo do desconto não deve figurar na estrutura principal do programa. Veja a figura 2.10 .

Figura 2.10 - Uso de sub-rotinas.

```
210 '
211 '   Desconto devido ao IAPAS
212 '
213 GOSUB 3000   ' Define o valor da
                   REFERENCIA e da
                   ALIQUOTA em funcao da
                   legislacao vigente.
214 IAPAS = REFERENCIA * ALIQUOTA
300 '
301 '                .
302 '                .
303 '                .
3000 '
3001 '  Rotina que calcula o valor da
3002 '  REFERENCIA e da ALIQUOTA.
3003 '
```

NUMERAÇÃO DAS INSTRUÇÕES

As únicas restrições do BASIC quanto aos números das instruções são as seguintes:

A - os números devem ser em ordem crescente.
B - os números devem estar compreendidos entre 0 e 65529.

Mesmo entre programas muito longos é difícil encontrar algum que utilize uma gama muito grande de números entre os limites acima.

Se você tem como hábito produzir programas, observará que em muitos deles, existem trechos que são exatamente iguais, independentemente da natureza do programa do qual eles fazem parte.

No exemplo acima, a rotina 3000 que determina o valor de REFERENCIA e o valor da ALIQUOTA em função do SALARIO é uma rotina que pode ser empregada em muitos programas, além do programa de Folha de Pagamento.

Se a tabela de cálculo for mantida em um arquivo em disco, a determinação do valor de REFERENCIA e a correspondente ALIQUOTA podem ser determinadas conforme a figura 2.11 .

Figura 2.11 - Lendo valores de arquivos em disco.

```
3000 '  --------------------------------
3001 '  Determina o valor de REFERENCIA e
3002 '  a ALIQUOTA em funcao do SALARIO.
3003 '  --------------------------------
3004 '
3005 OPEN "TABIAPAS" AS #1 LEN=12
3006 FIELD #1, 8 AS REF$,4 AS ALIQ$
3007 REG = 1
3008 GET #1,REG
3009 IF SALARIO < CVD(REF$) THEN
                  REFERENCIA = CVD(REF$)
                : ALIQUOTA   = CVS(ALIQ$)
                : CLOSE #1
                : RETURN
3010 REG = REG + 1
3011 IF NOT EOF(1) THEN GOTO 3008
3012 RETURN
```

A padronização da numeração de trechos de programas traz inúmeras vantagens ao programador.

Uma delas é a de fixar o conteúdo do trecho ao número. No exemplo acima, a numeração de 3000 a 3012 poderá ser utilizada sempre para essa rotina, em qualquer programa desenvolvido. Assim, quando o programador estiver analisando a listagem de um programa qualquer e encontrar a instrução:

```
GOSUB 3000
```

saberá, automaticamente, que se trata de uma rotina que determina o valor de REFERENCIA e da ALIQUOTA.

Outra vantagem, como veremos mais adiante, e.contramos na elaboração de um programa que necessita dessa mesma rotina. O comando MERGE fará com que a rotina seja automaticamente incorporada ao programa, sem que tenhamos que digitá-la novamente.

ESPAÇOS EM BRANCO

A velocidade em que uma instrução é executada pelo computador depende do seu comprimento. Instruções curtas são executadas em menor tempo.

Medindo-se o tempo gasto para executar as instruções da linha 105 do programa da figura 2.12,

observamos que existe um ganho de tempo aproximado de 1,81% quando os espaços em branco são eliminados, ou seja, a linha 105 é substituída por:

```
105 RES = SOMA + PARCIAL
```

Figura 2.12 - Programa com espaços em branco.

```
100 SOMA = 10
101 PARCIAL = 20
102 MX = 100
103 TIME = 0
104   FOR N=1 TO MX
105   RES = SOMA + PARCIAL
106   NEXT N
107 PRINT "TEMPO = "TIME
108 END
```

Espaços em branco facilitam a visualização e a compreensão da instrução e, de acordo com o que é feito pelo trecho do programa, essa pequena diferença de tempo pode não ser significativa. Muitas vezes é preferível admitir um prazo maior na execução para se ganhar rapidez na interpretação da listagem.

EDIÇÃO DA LISTAGEM

O comando LLIST é muito pobre em recursos. As listagens impressas obtidas com esse comando, não têm as páginas numeradas e não são respeitadas as margens de topo e de rodapé, tornando a listagem inadequada para arquivo.

Para contornar essas restrições, o programador poderá valer-se de um Editor de Textos. Nestes casos, o programa deve ser salvo no disco no formato de texto, o que é feito pela instrução:

```
SAVE "A:PROGRAMA.FNT",A
```

com o parâmetro "A" para indicar o nosso desejo de que o arquivo no disco esteja formatado em "ASCII". Após salvar o programa no disco, o programador deve carregar o Editor de Textos e digitar os comandos para a impressão do arquivo.

Caso o leitor não disponha de um Editor de Textos que atenda às necessidades de numerar as páginas e incluir linhas no topo e rodapé da listagem, poderá desenvolver um Listador, baseado no exemplo da figura 2.13 .

Figura 2.13 - Programa Listador

```
100 ' ==============================================
101 '   LISTADOR.TXT = Imprime a listagem de programas
102 ' ==============================================
103 '
104 '      Programa para imprimir a listagem de
105 '      programas, com numeracao de paginas e
106 '      cabecalho em cada pagina.
107 '
108 '      R.M.Watanabe - 24/02/87
109 '
200 '      ----------------
201 '      Entrada dos Dados
202 '      ----------------
203 '
204   PRINT "LISTADOR"
205    INPUT "Fornecer o Nome do Programa";N$
206     INPUT "Fornecer o Cabecalho das paginas";C$
207      INPUT "Fornecer a data da listagem";D$
211       E$ = SPACE$(6)
212        MAX = 50
213         P = 1
300 '
301 '      -----------------------
302 '      Procura o arquivo no disco
303 '      -----------------------
304 '
305   OPEN N$ FOR INPUT AS #1
306    L = 51
310     LINE INPUT #1,L$
311      IF L > MAX THEN GOSUB 1000
312       LPRINT E$,L$
313        L = L + 1
314         IF NOT EOF(1) THEN GOTO 310
316          PRINT "FINAL DA LISTAGEM"
317           CLOSE #1
318            END
1000 '
1001 ' Rotina cabecalho
1002 '
```

```
1003 IF P = 1 THEN GOTO 1008
1004      FOR I = 1 TO INT(66-MAX) - 4
1005          LPRINT
1006          NEXT I
1008      LPRINT C$" - "D$" - "P" - "
1009      LPRINT
1010      LPRINT
1011      LPRINT
1012      P = P + 1
1013      L = 1
1014      RETURN
```

Programas longos produzem listagens igualmente longas, consumindo um bom tempo para a impressão.

Ocorre, porém, que nem sempre estamos interessados em obter uma listagem completa do programa. Em situações de alterações, geralmente, apenas um pequeno trecho é alterado. Nestes casos, para efeito de arquivo, necessitamos imprimir apenas o trecho ou a página alterada.

PROGRAMA TEXTO E DE EXECUÇÃO

Um programa que possua uma lógica complexa poderá resultar em uma listagem em que a quantidade de comentários e espaços em branco são tantos que prejudicam sensivelmente a velocidade de execução.

Nestes casos, sugere-se operar com 2 tipos de programas: um, com todos os comentários e espaços em branco, que podemos chamar de PROGRAMA.TXT e outro em que os comentários e espaços em branco foram eliminados, e que podemos chamar de PROGRAMA.BAS.

O PROGRAMA.TXT será utilizado nas análises para alterações no programa, sendo executado apenas nos testes de implantação, enquanto que o PROGRAMA.BAS será utilizado nas execuções rotineiras, após a implantação.

As alterações que tiverem que ser introduzidas no programa devem, sempre, serem efetuadas e testadas no PROGRAMA.TXT.

Depois que as alterações forem testadas e aprovadas, o programador deve produzir o PROGRAMA.BAS. Para isso deve editar o programa, eliminando os comentários e os espaços em branco supérfluos.

Os programadores "profissionais" poderão automatizar essa produção com um programa do tipo apresentado na figura 2.14 .

Figura 2.14 - Eliminador de linhas REM.

```
100 ' ===============================================
102 ' TIRAREM.BAS = Limpa a listagem de um programa
104 ' ===============================================
106 ' Elimina os comentarios e os
108 ' espacos em branco superfluos
110 ' do programa
112 '
114 ' R.M.Watanabe                    19/05/87
200 ' ---------------------------------
202 ' Entrada do nome do programa
204 ' ---------------------------------
206   INPUT "nome do programa ";N$
208   OPEN N$ FOR INPUT AS #1
210 LINE INPUT #1,A$
212  B$ = " '":GOSUB 300
214   B$ = " REM":GOSUB 300
216    B$ = ":'":GOSUB 300
218     B$ = ":REM":GOSUB 300
220      I = INSTR(A$," ")
222       IF I <> 0 THEN A$ = LEFT$(A$,I-1) +
          RIGHT$(A$,LEN(A$)-I):GOTO 220
224        IF  LEN(A$) < 5 THEN GOTO 228
226         PRINT A$
228          IF NOT EOF(1) THEN GOTO 210
230           CLOSE #1
232            END
300 ' ---------------
302 ' Rotina Elimina
304 ' ---------------
306   I = INSTR(A$,B$)
308    IF I <> 0 THEN A$ = LEFT$(A$,I-1):GOTO 306
310     RETURN
```

O programa da figura 2.14 está apresentando a listagem na tela do vídeo. Faça uma alteração no mesmo para gravar o programa modificado no disco.

SISTEMA DE PROGRAMAS

O comprimento de um programa está diretamente relacionado aos fatores seguintes:

- a complexidade do problema.
- diversidade de opções oferecidas pelo programa.

Problemas complexos e variedade de opções oferecidas tendem a produzir listagens grandes e na utilização de tais programas o programador poderá defrontar-se com os seguintes problemas:

- Insuficiência de memória.
- Dificuldades na manipulação do programa.

Conforme apresentado no Capítulo 8, o espaço disponível para programas e variáveis está restrito a determinados limites e um programa grande poderá não caber na área disponível para o programa ou, mesmo que caiba, poderá fazer restar uma área insuficiente para a definição das variáveis do programa.

Nestes casos, a solução será a de segmentar o programa em vários outros menores, criando-se assim, um conjunto de programas mais conhecido como Sistema de Programas. Um sistema de programas poderá ser um único programa ou um conjunto de vários programas, desenvolvidos para resolver um determinado problema.

Os vários programas de um sistema devem estar ligados entre si mediante uma lógica estabelecida e a execução de um ou outro será comandada pelos demais programas do sistema.

Um programa pode acionar a execução de um outro por meio do comando RUN.

Para exemplificar isso, digite e grave os 3 programas da figura 2.15 .

Figura 2.15 - Segmentação de uma rotina.

```
100 ' PARTE 1
102 PRINT "PARTE 1"
104 RUN "PARTE2"

100 ' PARTE 2
102 PRINT "PARTE 2"
104 RUN "PARTE3"

100 ' PARTE 3
102 PRINT "PARTE 3"
104 END
```

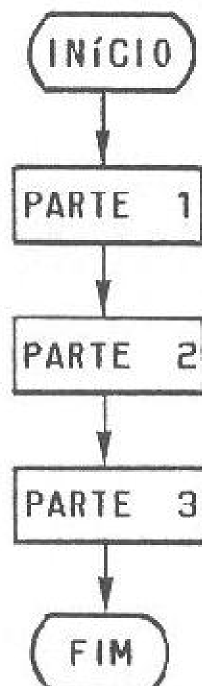
Salve os 3 programas com os nomes PARTE1, PARTE2 e PARTE3 respectivamente e execute o primeiro com a instrução:

```
RUN "PARTE1"
```

Existem muitas maneiras diferentes de se interligar os programas de um sistema.

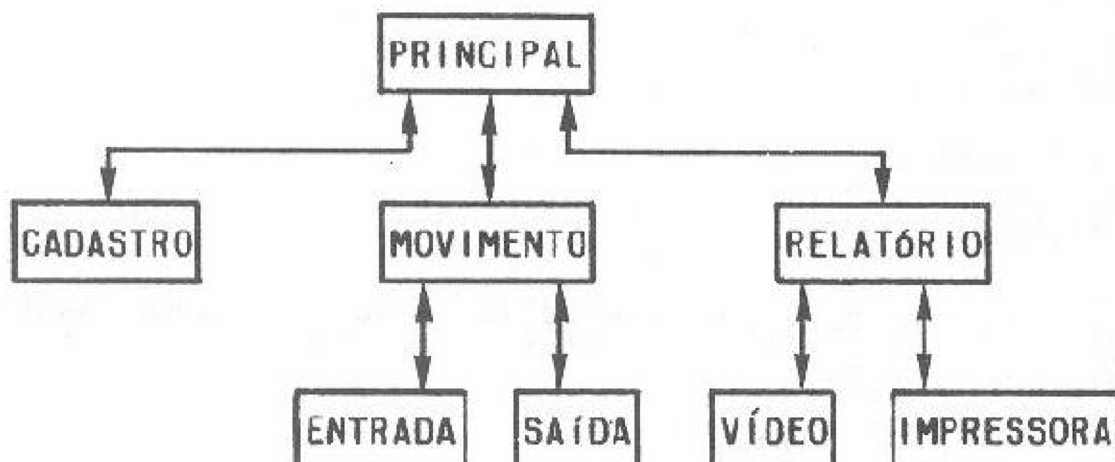
No exemplo acima os programas estão interligados linearmente, como apresentado na figura 2.16.

Figura 2.16 - Estrutura Linear.



A estrutura de interligação dos programas segue em geral, a estrutura em árvore, havendo um programa principal que executa os demais, podendo haver alguns desses que executam outros programas. Veja a figura 2.17.

Figura 2.17 - Estrutura em Árvore.



Nesses casos, a escolha do programa a ser carregado pelo programa PRINCIPAL será feita pelo usuário através de um cardápio de opções apresentado pelo programa PRINCIPAL.

O programa que foi carregado, ao finalizar a sua tarefa, poderá retornar ào programa que o carregou por meio de um comando RUN.

Maiores detalhes sobre a interligação de programas serão apresentados no capítulo 8.

RESPEITO AO USUÁRIO

O usuário é um ente impaciente e espera que o computador realize as tarefas instantaneamente.

Quem já efetuou a carga de programas a partir de uma fita cassete está habituado a esperar o tempo necessário para a transferência do programa da fita para a memória.

Mas nem todos os usuários passaram por essa experiência (ou prova de paciência) de modo que durante a fase de carga de um outro programa, é conveniente os programas apresentarem uma mensagem do tipo:

"Estou carregando um outro programa.
Aguarde alguns instantes."

A fim de facilitar a programação, o programador poderá desenvolver uma rotina do tipo apresentado na figura 2.18 .

Figura 2.18 - Rotina de Carga de Outro Programa.

```
5100 ' ------------------------------------
5102 ' ESPERE.FNT = Carrega Outro Programa
5104 ' ------------------------------------
5110 CLS
5112 PRINT TT$
5114 PRINT"------> TRANSFERENCIA <----------------"
5116 PRINT
5118 PRINT"Estou providenciando a carga do"
5120 PRINT"programa ";PP$
5122 PRINT
5124 PRINT"Favor aguardar alguns instantes."
5126 RUN PP$
```

Salve esta rotina para ser utilizada no sistema apresentado no Capítulo 9.

CUIDADO COM O NOME DO PROGRAMA

Durante a fase de digitação do programa e principalmente durante a fase de testes do mesmo, existe a necessidade de muitos LOAD e SAVE do programa.

O programador muito atarefado, que desenvolve um programa atrás do outro, poderá confundir, com relativa facilidade, o nome de um programa. Além disso, nomes muito compridos ou complexos costumam provocar erros de digitação, resultando em vários programas iguais com nomes um pouco diferentes e, no dia seguinte, após ter "varado" a madrugada a procura de um erro no programa, o programador terá uma certa dificuldade em identificar aquele que foi alterado por último.

Para evitar esse tipo de confusão, recomenda-se utilizar nomes curtos e, se possível, com apenas uma única letra. Os comandos abaixo são muito fáceis de serem digitados:

```
LOAD "P"

SAVE "P"
```

Habitue-se a sempre dar esse nome aos programas em desenvolvimento. Isso evitará muitos aborrecimentos.

Depois que o programa estiver totalmente pronto e inteiramente testado, bastará dar o nome definitivo com o comando:

```
NAME "P" AS "FOLPAG.BAS"
```

3

técnicas de entrada de dados

Toda vez que pressionamos alguma tecla do microcomputador, estamos querendo introduzir algum tipo de informação. Pode ser o nome de um funcionário para que o programa calcule a Folha de Pagamento, a taxa de juros para o programa calcular o valor das prestações, um número que representa uma das alternativas apresentadas por um cardápio de opções ou mesmo um comando para interromper a execução do programa.

Analisando as situações em que as teclas são pressionadas podemos constatar que a intenção com que isso é feito varia conforme as necessidades do programa e de fatos externos a ele, tais como erro de digitação ou a necessidade imprevista de interrupção do serviço.

O programador deve analisar os fatos segundo a ótica do programa e também segundo a do usuário.

Do ponto de vista do programa, essas situações podem ser agrupadas conforme o seguinte:

A - Fornecimento de dados ao programa.

São situações em que o programa solicita a informação que será processada, tais como o nome, o endereço, a idade, a quantidade, a taxa de juros, etc.

B - Escolha de opções.

São situações em que o programa, prevendo diversas alternativas de processamento, apresenta uma relação de opções, para que o usuário escolha uma delas.

C - Comandos para o programa.

São situações em que desejamos passar determinados comandos para o programa.

Vejamos o que pode ocorrer do ponto de vista do usuário:

A - Durante a digitação de uma determinada informação, foi cometido um engano, detectado antes de se terminar a digitação da mesma.

B - Foi fornecida uma informação errada que só foi notada ao se digitar a informação seguinte.

C - Durante a digitação de uma série de dados, ouve uma interrupção no trabalho para que o usuário pudesse, por exemplo, atender ao telefone, e quando retornou à atividade, não conseguiu lembrar em que item a digitação foi interrompida.

D - Foi escolhida uma opção errada, constatando-se essa falha, quase ao mesmo tempo em que a tecla da opção foi pressionada.

Erros de digitação são muito comuns. Um programa "profissional" deve, até certo ponto, permitir algum tipo de erro e oferecer ao usuário a oportunidade de corrigir o mesmo.

Vejamos como é possível conciliar as diversas necessidades do programa com as situações, muitas vezes imprevisíveis, que cercam o usuário quando ele for digitar os dados.

COMANDOS E FUNÇÕES PARA A ENTRADA DE DADOS

Os comandos e funções que a linguagem BASIC oferece para a entrada de dados pelo teclado são os seguintes: CSRLIN, INKEY$, INPUT, INPUT$, LINE INPUT, POS.

Vamos analisar cada um dos comandos e verificar até que ponto ele poderá ser útil nas situações que o programa vai enfrentar para recebimento de dados.

CSRLIN - Fornece a linha do cursor.

Esta função nos fornece o número da linha em que está posicionado o cursor. Os valores possíveis situam-se entre 0 e 23.

Pode ser útil em situações de escolha de uma das alternativas apresentadas por um menu de opções.

Exemplo: O programa da figura 3.1 apresenta uma tela com 5 opções, onde o usuário deve deslocar o cursor até a opção desejada, com o auxílio das teclas ▼ e ▲ e, em seguida, pressionar a tecla RETURN.

Figura 3.1 - Uso de CSRLIN.

```
100 CLS
101 PRINT
102 PRINT
103 PRINT "A - Opcao 1"
104 PRINT
105 PRINT "B - Opcao 2"
106 PRINT
107 PRINT "C - Opcao 3"
108 PRINT
109 PRINT "D - Opcao 4"
110 PRINT
111 PRINT "E - Opcao 5"
112 PRINT
113 PRINT
114 PRINT "Escolha  uma  das opcoes
              acima e pressione RETURN"
115 HC = 15
116 VC = 2
117 LOCATE HC,VC : PRINT "<---"
118 I$ = INKEY$
119 IF I$="" THEN GOTO 118
120 IF ASC(I$)=13 THEN GOTO 130
121 LOCATE HC,VC : PRINT SPC(4)
122 IF ASC(I$)<>30 THEN GOTO 126
123 VC = VC - 2
124 IF VC<2 THEN VC = 2
125 GOTO 117
126 IF ASC(I$)<>31 THEN GOTO 117
```

```
127 VC = VC + 2
128 IF VC>10 THEN VC = 10
129 GOTO 117
130 CR = INT(CSRLIN/2)
131 LOCATE ,16 : PRINT "Voce escolheu a opcao ";CR
132 END
```

Comentários:

100	Apaga toda a tela para a apresentação das opções.
101-111	Apresenta as 5 opções.
112-114	Instrui o usuário de como proceder.
115-116	Define um valor inicial para as variáveis HC e VC que vão comandar o posicionamento da seta indicativa.
117	Posiciona o cursor na coordenada HC,VC e desenha a seta indicativa. Na primeira vez, o posicionamento será na coordedenada definida nas instruções 115 e 116. Nas outras vezes, o posicionamento será na coordenada redefinida em 123 ou 127.
118-119	Aguarda que alguma tecla seja pressionada. O valor da tecla é armazenado na variável I$ para que possa ser comparado em 120, 122 e 126.
120	Verifica se a tecla pressionada foi a tecla RETURN, que possui o código ASCII igual a 13. Se sim, sai do ciclo, pulando para 130.
121	Reposiciona o cursor na coordenada HC, VC onde havia sido desenhada a seta indicativa e apaga-a, imprimindo espaços em branco.
122	Verifica se a tecla pressionada foi a tecla ▲, que possui o código ASCII igual a 30.
123	Se sim, diminui o valor da ordenada VC.
124	Limita o menor valor da ordenada VC em 2.
125	Volta para repetir o ciclo.
126	Verifica se a tecla pressionada foi a tecla ▼, que possui o código ASCII igual a 31.
127	Se sim, incrementa o valor da ordenada VC.
128	Limita o maior valor da ordenada VC em 10.
129	Volta para repetir o ciclo.
130	Calcula o valor da opção escolhida, com base na posição do cursor, fornecida pela função CSRLIN.
131	Imprime o valor calculado.
132	Encerra a execução do programa.

INKEY$ - Obtém o caractere da tecla pressionada.

Esta função fornece o valor da tecla pressionada no instante em que o programa executa a função.

É muito difícil o usuário acertar o momento exato em que a função está sendo executada. Por isso, é usual colocarmos a função dentro de um ciclo, verificando constantemente se o valor da tecla "pressionada" é uma string vazia ou não. Enquanto nenhuma tecla for pressionada, o valor da função será uma string vazia. Veja as linhas 100 e 101 da figura 3.2 .

A grande vantagem dessa função reside no fato de ela aceitar qualquer tecla ou combinação com as teclas CTRL, SHIFT, LGRA (GRAPH) e RGRA (CODE).

A título de ilustração apresentamos na figura 3.2 um programa que fornece os códigos ASCII para cada tecla ou combinação de teclas pressionadas.

Figura 3.2 - Uso de INKEY$.

```
100 I$ = INKEY$
101 IF I$="" THEN GOTO 100
102 PRINT ASC(I$)
103 GOTO 100
```

Observe que quase todos os códigos ASCII desde o 0 até o 255 são possíveis de serem obtidos pelo teclado. Observe também que o tradicional CTRL-C que interrompe a execução de programas, não tem o mesmo efeito neste caso.

INPUT - Introduz o valor de uma variável através do teclado.

Este comando interrompe a execução do programa e permite que novos dados sejam fornecidos ao programa, através do teclado.

Dos comandos de entrada de dados, este é o mais empregado nos programas, pois permite:

- Antepor uma frase para orientar o usuário quanto ao dado, seu tipo ou unidade em que o mesmo deve ser fornec'‘o.

```
100 INPUT "Area do local em m2";AREA
```

- Aquilo que é digitado no teclado é impresso na tela do vídeo, permitindo a edição da informação.

- Os dados numéricos são converti os e armazenados diretamente nas variáveis.

- Diversas informações diferentes podem ser introduzidas por apenas uma única instrução.

A execução do programa da figura 3.3 deverá produzir algo como o apresentado na figura 3.4 .

Figura 3.3 - Uso de INPUT.

```
100 INPUT "Nome, Idade, Peso e Bairro";N$,I,P,B$
101 PRINT "nome   = ";N$
102 PRINT "idade  =";I
103 PRINT "peso   =";P
104 PRINT "bairro = ";B$
```

Figura 3.4 - Tela Obtida Após o Uso do Programa INPUT.

```
run
Nome, Idade, Peso e Bairro? Carlos,23,
60,Tatuape
nome   = Carlos
idade  = 23
peso   = 60
bairro = Tatuape
Ok
```

O uso deste comando em programas "profissionais" apresenta algumas restrições operacionais dificultando, em muito, a digitação das informações. Vejamos algumas delas.

A vírgula não pode figurar como parte da informação.

O comando:

```
100 INPUT "Endereco";A$
```

deveria, do ponto de vista do usuário, ser respondido com:

```
Endereco? Av. Brasil, 4.587
```

Porém, ao se pressionar a tecla RETURN, será

apresentada a seguinte mensagem de erro:

?Extra ignored

Indicando que o computador considerou que foram digitadas duas informações distintas, como se estivesse executando uma instrução do tipo:

100 INPUT "Endereco";A$,B$

quando a informação "Av. Brasil" seria associada à variável A$ e a outra informação "4.567" seria associada à variável B$.

Porém, o programa está esperando a entrada de apenas uma única informação. Daí a resposta do computador, dizendo que a informação extra, ou adicional, não foi considerada.

Outra restrição importante, deste comando, é que ele só aceita informações compostas apenas com os caracteres que tenham uma representação gráfica na tabela ASCII. Os códigos que não possuam uma representação gráfica, tais como o TAB, o ESC, o espaço em branco no início da informação e todos os códigos gerados em combinação com a tecla CTRL são, normalmente, desprezados pelo comando INPUT.

Desse modo, fica difícil a programação de situações em que o usuário percebeu, depois que a tecla RETURN foi pressionada, que uma determinada informação foi digitada com erros e deseja voltar a um estágio anterior para poder corrigir o dado que foi fornecido.

<u>INPUT$</u> - Lê uma quantidade especificada de caracteres introduzidos pelo teclado.

Esta função, parecida com o comando INPUT, permite que uma informação de comprimento determinado seja fornecida ao programa.

Embora parecida com o comando INPUT, apresenta as seguintes diferenças:

- Não há a necessidade de se pressionar a tecla RETURN ao final da informação. Isso é muito bom, pois é uma tecla a menos que o usuário terá que digitar, mas obriga a que todos os dados tenham o mesmo comprimento. A título de exemplo, veja o programa da figura 3.5 .

Figura 3.5 - Uso do comando INPUT$.

```
100 PRINT "Sigla do Estado=";
101 ES$ = INPUT$ (2)
102 PRINT
103 PRINT "Foi digitado: ";ES$
```

- O comando aceita todos os códigos da tabela ASCII, mesmo aqueles que não possuem representação gráfica, tais como a tecla TAB, ESC e os códigos gerados pela associação com a tecla CTRL. O programa da figura 3.6 fornece os mesmos códigos ASCII que o programa apresentado na função INKEY$ (figura 3.2).

Figura 3.6 - Uso do INPUT$.

```
100 I$ = INPUT$ (1)
101 PRINT ASC(I$)
102 GOTO 100
```

- Não é possível a colocação de alguma mensagem para orientar o usuário. Nos casos em que a mensagem é imprescindível, o programador deve utilizar o comando PRINT.

- As teclas digitadas não são apresentadas no vídeo, de modo que se torna quase impossível a digitação de dados, exceto em casos de digitação de senhas, quando se deseja justamente isso.

<u>LINE INPUT</u> - Associa um dado introduzido pelo teclado a uma variável string.

Esse comando interrompe a execução do programa e permite que novos dados sejam fornecidos ao programa, através do teclado, como no comando INPUT.

As diferenças, em relação ao comando INPUT são as seguintes:

- Somente uma única informação pode ser fornecida a cada comando. Caso a instrução a seguir seja executada, será produzido um Erro de Sintaxe.

```
100 LINE INPUT "Nome e Idade"; N$,I$
```

- A vírgula pode fazer parte da informação. O programa da figura 3.7, quando executado, deve produ-

zir o resultado da figura 3.8 .

Figura 3.7 - Uso do comando LINE INPUT.

```
100 LINE INPUT "Endereço="; E$
101 PRINT E$
```

Figura 3.8

```
Endereço= Av.Brasil,4.567
Av. Brasil, 4.567
```

Diferentemente do comando INPUT, neste caso, a informação completa, incluindo-se a vírgula, é associada à variável A$.

- As informações são tratadas, sempre, como strings, de modo que as informações numéricas devem ser convertidas pelo programa. Veja o programa da figura 3.9 .

Figura 3.9 - Conversão de strings para números.

```
100 LINE INPUT "Taxa de juros=";T$
101 TJ = VAL(T$)
```

- O comando só aceita os caracteres que possuem representação gráfica na tabela ASCII, não aceitando as teclas do tipo ESC, TAB e os códigos gerados em combinação com a tecla CTRL.

- Aceita os espaços em branco digitados no início da informação.

<u>POS</u> - Indica a abcissa X que o cursor ocupa.

Esta função nos fornece o número da coluna em que está posicionado o cursor. Os valores possíveis situam-se entre 0 e 39.

Pode ser útil em situações de escolha de uma das alternativas apresentadas por um menu de opções.

Exemplo: O programa da figura 3.10 apresenta uma tela com 4 opções, onde o usuário deve deslocar o cursor até a opção desejada, com o auxílio das teclas do cursor, em seguida, pressionar a tecla RETURN.

Figura 3.10 - Uso da função POS.

```
100 CLS
101 PRINT "Opcao 1 -----------+"
102 PRINT "                   !"
103 PRINT "Opcao 2 --------+  !"
104 PRINT "                !  !"
105 PRINT "Opcao 3 -----+  !  !"
106 PRINT "             !  !  !"
107 PRINT "Opcao 4 --+  !  !  !"
108 PRINT "          !  !  !  !"
109 PRINT "          A  B  C  D"
110 CR = 1
111    LOCATE 7+3*CR, 9:PRINT "^"
112    LOCATE 7+3*CR,10:PRINT "!"
113    LOCATE 7+3*CR,11:PRINT "!";
114 I$ = INPUT$ (1)
115 IF ASC (I$) = 13 THEN GOTO 127
116    LOCATE 7+3*CR, 9:PRINT " "
117    LOCATE 7+3*CR,10:PRINT " "
118    LOCATE 7+3*CR,11:PRINT " "
119 IF ASC (I$) <> 28 THEN GOTO 123
120 CR = CR + 1
121 IF CR > 4 THEN CR = 4
122 GOTO 111
123 IF ASC (I$) <> 29 THEN GOTO 111
124 CR = CR - 1
125 IF CR < 1 THEN CR = 1
126 GOTO 111
127 CR = INT((POS(0)-7)/3)
128 LOCATE 0,14
129 PRINT "Voce escolheu a opcao";5-CR
130 END
```

Comentários:

100 — Apaga toda a tela para a apresentação das opções.

101-109 — Apresenta as 4 opções.

110 — Define um valor inicial para a variável CR, que vai indicar o número da opção e será empregada junto com o comando LOCATE para posicionar o cursor.

111-113 — Desenha a seta que indica uma das opções.

114 — Aguarda que seja pressionada alguma tecla e armazena o código da tecla pressionada na variável I$.

115	Verifica se a tecla pressionada é a tecla RETURN.
116-118	Apaga a seta indicativa, imprimindo brancos.
119	Verifica se a tecla pressionada foi a tecla →, cujo código ASCII é 28.
120	Incrementa a variável CR.
121	Verifica se a variável CR está dentro do limite superior.
122	Volta para repetir o ciclo.
123	Verifica se a tecla pressionada é a tecla ←, cujo código ASCII é 29.
124	Decrementa a variável CR.
125	Verifica se a variável CR está dentro do limite inferior.
126	Volta para repetir o ciclo.
127	Obtém o valor da opção escolhida a partir da posição do cursor, obtida com a função POS.
128	Posiciona o cursor.
129	Imprime o número da opção escolhida.
130	Encerra a execução do programa.

ROTINA PARA A ENTRADA DE DADOS

Os comandos e funções que permitem a entrada de informações pelo teclado apresentam características próprias de funcionamento.

De forma resumida, tais características são as seguintes:

CSRLIN - Fornece apenas a posição vertical do cursor, não serve para a digitação de dados.

INKEY$ - Aceita qualquer caractere do teclado, porém não mostra no vídeo a tecla pressionada.

INPUT - Não identifica os códigos produzidos em combinação com a tecla CTRL e não aceita a vírgula no meio da informação. Mostra no vídeo as teclas pressionadas.

INPUT$ - A informação deve ter um comprimento fixo, e o que for digitado não é mostrado no vídeo.

LINE INPUT - Aceita a vírgula, porém não reconhece os códigos produzidos em combinação com a tecla CTRL. Mostra no vídeo, as teclas pressionadas.

POS - Fornece apenas a posição horizontal do cursor, não serve para a digitação de dados.

A escolha do comando adequado para a entrada de dados, depende do tipo de informação que será digitada para o programa.

As teclas, sozinhas ou combinadas com outras do tipo SHIFT, CTRL, CODE ou GRAPH, (ou RGRA e LGRA), produzem códigos que são enviados pelo teclado para a memória do computador.

Tais códigos são combinações de 8 bits, com significados padronizados pela convenção ASCII.

Embora o teclado tenha a capacidade de gerar 256 códigos distintos, nem sempre desejamos, ou necessitamos, de todos eles.

Vejamos, para cada situação de digitação de informações, as faixas de códigos que possam interessar aos programas.

Na digitação de Dados Numéricos, estaremos fornecendo ao programa, informações compostas pela combinação dos algarismos 0 a 9 e símbolos + - . , . Observando a tabela ASCII, notamos que esses símbolos correspondem aos códigos &H2B a &H39.

Na digitação de Dados Alfabéticos, somente maiúsculas, estaremos fornecendo ao programa informações compostas pela combinação das letras de A a Z, que correspondem, na tabela ASCII, aos códigos de &H41 a &H5A.

Na digitação de Dados Alfabéticos, inclusive minúsculas, estaremos fornecendo ao programa informações compostas pela combinação das letras de A a z, que correspondem, na tabela ASCII, aos códigos de &H41 a &H7A.

Na Escolha de Opções, usamos apenas as teclas ▲ e ▼ para posicionar o cursor sobre a opção desejada e mais a tecla RETURN para indicar a escolha. Neste caso, os códigos que nos interessam são os seguintes:

&H1E para a tecla ▲
&H1F para a tecla ▼
&H0D para a tecla RETURN

Como se vê, em cada situação de digitação de informação, a quantidade de códigos distintos é limitada e bem inferior aos 256 possíveis de serem gerados pelo teclado.

Embora as situações de digitação de informações sejam distintas e possam até ser representadas

por telas distintas (uma tela só para a Escolha de Opções, outra para a Digitação dos Dados e outra para dar comandos para o programa), seria interessante, misturar determinadas teclas de controle, possibilitando a digitação de comandos em situações em que o programa espera a digitação de algum dado do problema.

Um exemplo disso seria na digitação da Área do Terreno em um programa de Topografia. A tela de digitação da Área poderia ser algo como o apresentado na figura 3.11 ou algo mais completo como o ilustrado na figura 3.12 .

Figura 3.11 - Programa de topografia.

```
        PROGRAMA DE TOPOGRAFIA
AREA = _
```

Figura 3.12 - Programa de topografia.

```
        PROGRAMA DE TOPOGRAFIA
Fornecer a área do terreno em m²: _
```

Para o usuário que estiver utilizando o programa pela primeira vez, a primeira tela poderá provocar uma dúvida quanto à unidade em que a Área deva ser fornecida ao programa.

Para o usuário que já estiver familiarizado com o programa, a segunda tela poderá provocar uma fadiga visual pelo excesso de informações apresentadas na tela do vídeo. Além disso, em programas cuja Entrada de Dados seja constituída por um conjunto grande de dados, a indicação completa do tipo de informação em cada campo, poderá fazer com que os dados necessários não caibam em uma única tela, dificultando a sua digitação.

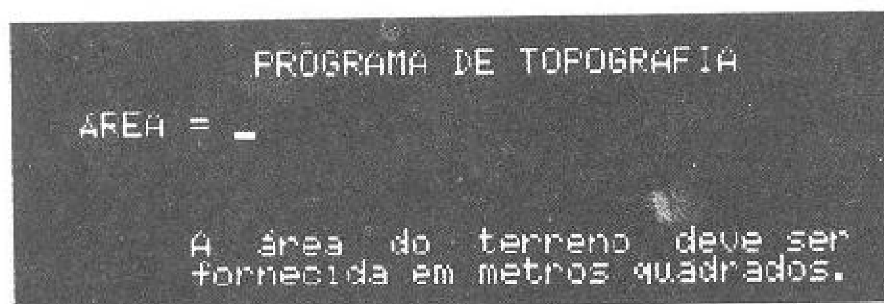
Uma das maneiras de contornar essas situações seria o emprego de uma tela simples e a menção da unidade da Área no manual do programa e obrigar os novos usuários a lerem o manual.

Outra maneira, mais elegante, é introduzir uma tecla especial, por exemplo CTRL+A (A de auxílio) que, quando pressionada, com o cursor em qualquer dos

campos de Entrada de Dados, fará com que apareça, num canto da tela, uma pequena explicação sobre o tipo de dado esperado pelo programa naquele campo.

No exemplo anterior, após a pressão da tecla CTRL-A, a tela do vídeo se mostraria de acordo com a figura 3.13 .

Figura 3.13 - Tela com explicações.



Desse modo, a tela de digitação de dados fica mais enxuta, contendo apenas as explicações mínimas necessárias, facilitando a sua visualização. As demais explicações adicionais que eventualmente possam vir a ser necessárias não são normalmente apresentadas no vídeo, aparecendo somente quando a tecla CTRL-A for pressionada.

Como conjugar a entrada de dados com a digitação de comandos para o programa?

Pode-se elaborar uma Rotina de Entrada, e fazer com que em todas as situações de entrada de dados ou de digitação de comandos para o programa, a entrada seja feita por essa rotina.

Vejamos quais seriam as especificações para essa rotina:

1 - Permitir a entrada dos dados que representam a vida real, tais como o nome de pessoas, endereços, números e outros.

2 - Permitir que em situações em que o tamanho da informação seja previamente conhecido não seja necessário pressionar a tecla RETURN. Por exemplo, as siglas dos Estados brasileiros são formadas por 2 letras.

3 - Permitir que teclas especiais e de controle sejam pressionadas e devolver ao programa o código da tecla pressionada.

Para poder obedecer a essas especificações, iremos desenvolver uma rotina baseada no comando INKEY$, por ser o comando que abrange toda a faixa dos códigos ASCII.

1 - O núcleo básico da rotina é formado pela instrução:

320 K$ = INKEY$: IF K$="" THEN GOTO 320

quando o computador aguarda até que uma tecla qualquer seja pressionada.

2 - Logo que uma tecla for pressionada, a rotina deve verificar se essa tecla é ou não uma tecla de controle, representada por um códigos ASCII entre 0 e 31, com a instrução:

330 IF ASC(K$) < 32 THEN RETURN

3 - Não sendo uma tecla de controle, o símbolo digitado deve ser adicionado àquele anteriormente digitado, e armazenado na string A$:

340 A$ = A$ + K$

4 - A tecla pressionada deve ser impressa no vídeo:

345 PRINT K$;

5 - A string A$, sendo do tipo acumulativa, deve ser "zerada" no início da rotina:

300 A$ = ""

6 - Verificar se a string de entrada A$ já atingiu o comprimento esperado para a informação. O comprimento máximo da string deve ser previamente fornecido na variável CS:

350 IF LEN(A$) = CS THEN RETURN

7 - Não tendo, ainda, atingido o comprimento esperado, a rotina deve voltar para a recepção de mais uma tecla:

360 GOTO 320

8 - A rotina deve permitir a correção de algum caractere que tenha sido digitado erradamente. A indicação de correção pode ser feita com a tecla Back Space cujo código ASCII é 8:

```
328 IF ASC(K$)=8 THEN PRINT CHR$(8)" "CHR$(8);
                       :A$=LEFT$(A$,LEN(A$)-1)
                       :GOTO 320
```

9 - A rotina deve verificar se foi pressionada a tecla RETURN para finalizar o dado. A tecla RETURN possue o código 13:

```
324 IF ASC(K$) = 13 THEN RETURN
```

A rotina é apresentada na figura 3.14 .

Figura 3.14 - Rotina para checar RETURN.

```
300 A$ = ""
320 K$ = INKEY$ : IF K$="" THEN GOTO 320
324 IF ASC(K$) = 13 THEN RETURN
328 IF ASC(K$) = 8 THEN PRINT CHR$(8)" "CHR$(8);
                     :A$=LEFT$(A$,LEN(A$)-1)
                     :GOTO 320
330 IF ASC(K$) < 32 THEN RETURN
340 A$ = A$ + K$
345 PRINT K$;
350 IF LEN(A$) = CS THEN RETURN
360 GOTO 320
```

É claro, que as especificações da rotina não são somente aquelas acima apresentadas. O programador deverá, em função daquilo que ele desejar, elaborar suas próprias especificações.

Por enquanto, para sedimentar as idéias apresentadas, teste o funcionamento da rotina com um programa do tipo apresentado na figura 3.15 .

As teclas de controle, devem, na medida do possível, estar relacionadas com as situações práticas que podem ocorrer.

Figura 3.15 - Teste da rotina para checar RETURN.

```
100 CLS
101 PRINT "PROGRAMA TESTE"
102 LOCATE  0,10 : PRINT "AREA:"
103 LOCATE 6,10,1
104 CS=7
105 GOSUB 300
106 LOCATE 0,20 : PRINT STRNG$(40," ")
107 IF A$="" THEN IF ASC(K$)<32 THEN LOCATE 0,20
                               :PRINT "FOI DIGITADO:CONTROL -"
                               CHR$(ASC(K$)+64)";"
108 IF A$<>"" THEN LOCATE 0,20
                               :PRINT "FOI DIGITADO:"A$";"
109 GOTO 103
```

Assim, sugere-se adotar a seguinte convenção:

Ctrl+A	Auxílio. Apresenta uma orientação de como proceder.
Ctrl+C	Posiciona o cursor no último campo de digitação.
Ctrl+E	Posiciona o cursor no primeiro campo de digitação.
Ctrl+H	Move o cursor para o campo anterior.
Ctrl+L	Apaga o resto à direita do cursor.
Ctrl+Q	Move o cursor para cima.
Ctrl+U	Move o cursor para o campo seguinte.
Ctrl+X	Avança meia página.
Ctrl+Y	Limpa todos os campos de digitação.
Ctrl+Z	Move o cursor para baixo.

Analise as funções que as teclas CTRL desempenham em outros programas, inclusive aqueles que foram adquiridos prontos e procure padronizar as funções dessas teclas, para facilitar a vida do digitador.

Na escolha das teclas de controle, procure levar em consideração a disposição das teclas no teclado. As teclas CTRL+A, por exemplo, podem ser acionadas com uma só mão, enquanto que as teclas CTRL+P só podem ser acionadas com ambas as mãos.

Procure também, usar a mesma convenção de teclas em todos os seus programas, pois é terrível a digitação de dados quando em um programa CTRL+S significar uma coisa e em outro programa o mesmo CTRL+S significar coisa bem diferente.

EXEMPLO PRÁTICO

Vamos desenvolver um exemplo prático para a digitação dos dados.

A tela que deve ser usada para a digitação dos dados deve ter o aspecto da figura 3.16 .

Os campos para a digitação de dados são apresentados na tabela da figura 3.17 .

Com relação às teclas de controle a serem utilizadas no programa, são apresentadas juntamente com sua descrição na tabela da figura 3.18 .

A figura 3.19 mostra os códigos ASCII das teclas de controle utilizadas no programa.

Finalmente, a listagem completa do programa exemplo é apresentada na figura 3.20 .

Figura 3.16 - Tela para digitação dos dados.

```
    1234567890123456789012345678901234567890
   +----------------------------------------+
01 |                                        |
02 |           ## CADASTRAMENTO ##          |
03 |                                        |
04 |Data:XX/XX/XX                           |
05 |                                        |
06 |Nome:XXXXXXXXXXXXXXXXXXXXXXXXX          |
07 |End.:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     |
08 |                                        |
09 |Cidade:XXXXXXXXXXXXXXXXXXXX Estado:XX   |
10 |                                        |
   +----------------------------------------+
    1234567890123456789012345678901234567890
```

Figura 3.17 - Campos para digitação.

SEQUÊNCIA	FINALIDADE	TAMANHO	NOME DO CAMPO
1	Dia do mes	2	DIA$
2	Número do mes	2	MES$
3	Número do ano	2	ANO$
4	Nome	25	NOM$
5	Endereço	30	EDR$
6	Nome da cidade	20	CID$
7	Sigla do estado	2	EST$

Figura 3.18 - Teclas de controle.

TECLA	FUNÇÃO
CTRL + A	Auxílio.
CTRL + P	Próximo - avança para o próximo registro.
CTRL + L	Limpeza.
RETURN	Finaliza a entrada do dado.
BACK SPACE	Volta o cursor apagando.
SETAS	Movimentam o cursor.
HOME	Leva o cursor para o primeiro campo.
ESC	Apaga o campo onde está o cursor.
?	Mostra as teclas de controle.

A combinação de teclas CTLR+A (Auxílio) faz surgir uma explicação sobre o tipo de dado que deve ser digitado no campo onde está o cursor.

A pressão simultânea das teclas CTRL+P (Próximo) finaliza a entrada de dados, mostra um resumo do que foi digitado, limpa os campos de digitação de dados e posiciona o cursor no segundo campo (Nome) para a digitação dos dados da próxima pessoa. O primeiro campo é saltado, para não ter que se digitar a data novamente.

Em um programa completo, após as teclas CTRL+P, seria providenciada a gravação dos dados digitados em um arquivo em disco. No presente programa, onde procura-se ilustrar apenas a parte de digitação de dados, o programa irá mostrar, em um outra tela, todos os dados digitados.

A combinação CTRL+L (Limpeza) limpa os campos de digitação de dados e posiciona o cursor no primeiro campo.

A tecla RETURN finaliza a entrada do dado e posiciona o cursor no próximo campo.

A tecla BACK SPACE faz o cursor voltar para a coluna anterior apagando o caractere que ali se encontrar.

As teclas ► e ▼ posicionam o cursor no próximo campo. Caso o cursor esteja no último campo da tela será deslocado para o primeiro campo.

As teclas ◄ e ▲ posicionam o cursor no campo anterior. Caso o cursor esteja no primeiro campo da tela, não haverá movimento.

HOME posiciona o cursor no primeiro campo da tela.

Quando o cursor estiver no meio de um campo, ESC apaga o que foi digitado e posiciona o cursor no início do campo. Quando o cursor estiver no primeiro campo, volta à tela anterior. Como não há tela anterior, finaliza a execução do programa.

Figura 3.19 - Códigos ASCII das teclas de controle.

TECLA DE CONTROLE		CÓDIGOS MSX	IBM PC
CTRL+A	Auxílio	1	1
CTRL+P	Próximo	16	16
CTRL+L	Limpeza	12	12
RETURN	Finaliza	13	13
◄◄	Esquerda	8	8
►	Direita	28	077
▼	Próximo	31	080
◄	Esquerda	29	075
▲	Anterior	30	072
HOME	Início	11	071
ESC	Esqueça	27	27
?	Teclas	63	63

Figura 3.20 - Listagem do programa-exemplo.

```
100 ' ======================================
101 ' EXEMPLO.BAS = Testa Rotina de Entrada
102 ' ======================================
103 '
104 ' R.M.Watanabe                  20/03/87
105 '
200 ' --------------------
201 ' Parametros Iniciais
202 ' --------------------
210 DIM HC(7),VC(7),ZC(7)
212     HC(1)= 5:VC(1)=4:ZC(1)= 2
214     HC(2)= 8:VC(2)=4:ZC(2)= 2
216     HC(3)=11:VC(3)=4:ZC(3)= 2
218     HC(4)= 5:VC(4)=6:ZC(4)=25
220     HC(5)= 5:VC(5)=7:ZC(5)=30
222     HC(6)= 7:VC(6)=9:ZC(6)=20
224     HC(7)=35:VC(7)=9:ZC(7)= 2
230 DIA$ = SPACE$(2)
232  MES$ = SPACE$(2)
234   ANO$ = SPACE$(2)
```

```
236    NOM$ = SPACE$(25)
238     EDR$ = SPACE$(30)
240      CID$ = SPACE$(20)
242       EST$ = SPACE$(2)
244        CR = 1
250         GOTO 400
300 A$ = "":LOCATE ,,1
320 K$ = INKEY$:IF K$="" THEN GOTO 320
322 IF ASC(K$)= 13 THEN RETURN
324 IF ASC(K$)= 8  THEN PRINT CHR$(8)" "CHR$(8);:
                        A$=LEFT$(A$,LEN(A$)-1):
                        GOTO 320
326 IF ASC(K$) = 63 THEN A$=K$:RETURN
330 IF ASC(K$) <32 THEN A$=K$:RETURN
340 A$ = A$ + K$
342 PRINT K$;
350 IF LEN(A$) = CS THEN RETURN
352 GOTO 320
400 ' ----------------
401 ' Desenha a Tela
402 ' -------------
410   CLS
411   PRINT
412   PRINT SPC(10)"** CADASTRAMENTO **"
413   PRINT
414   PRINT
415   PRINT "Data:  /  /  :"
416   PRINT
417   PRINT "Nome:"SPC(25)":"
418   PRINT "End.:"SPC(30)":"
419   PRINT
420   PRINT "Cidade:"SPC(20)":Estado:  :"
500 ' ------------
502 ' Mostra Valores
504 ' -------------
510   LOCATE HC(1),VC(1):PRINT DIA$
512   LOCATE HC(2),VC(2):PRINT MES$
513   LOCATE HC(3),VC(3):PRINT ANO$
514   LOCATE HC(4),VC(4):PRINT NOM$
515   LOCATE HC(5),VC(5):PRINT EDR$
516   LOCATE HC(6),VC(6):PRINT CID$
517   LOCATE HC(7),VC(7):PRINT EST$
600 ' -------------
602 ' Entrada de Dados
604 ' ---------------
610 LOCATE HC(CR),VC(CR)
612 CS = ZC(CR):GOSUB 300
640    IF LEN(A$) = 0 THEN GOTO 710
```

```
650     IF ASC(A$) =  1 THEN ON CR GOSUB 1110,1120,
                             1130,1140,1150,1160,1170:
                             GOTO 600
652     IF ASC(A$) = 16 THEN GOTO 800
654     IF ASC(A$) = 12 THEN GOTO 230
656     IF ASC(A$) = 13 THEN GOTO 710
658     IF ASC(A$) =  8 THEN GOTO 720
660     IF ASC(A$) = 28 THEN GOTO 710
662     IF ASC(A$) = 31 THEN GOTO 710
664     IF ASC(A$) = 29 THEN GOTO 720
666     IF ASC(A$) = 30 THEN GOTO 720
668     IF ASC(A$) = 11 THEN CR=1:GOTO 600
670     IF ASC(A$) = 27 THEN ON CR GOSUB 1210,1220,
                             1230,1240,1250,1260,1270:
                             GOTO 500
672     IF ASC(A$) = 63 THEN GOTO 1300
674     IF ASC(A$) < 32 THEN GOTO 600
676 ON CR GOSUB 730,740,750,760,770,780,790
678 GOTO 710
710 ' ------> Avanca o Cursor
712             CR = CR + 1
714             IF CR > 7 THEN CR = 7
716             GOTO 600
720 ' ------> Retrocede o Cursor
722             CR = CR - 1
724             IF CR < 1 THEN CR = 1
726             GOTO 600
730 DIA$ = A$
732       RETURN
740 MES$ = A$
742       RETURN
750 ANO$ = A$
752       RETURN
760 NOM$ = A$
762       RETURN
770 EDR$ = A$
772       RETURN
780 CID$ = A$
782       RETURN
790 EST$ = A$
792       RETURN
800 ' ------> Finalizacao da Entrada
810 CLS
812  PRINT
814   PRINT
816    PRINT "FORAM DIGITADOS:"
818     PRINT
820      PRINT "DATA="DIA$"/"MES$"/"ANO$
```

```
822       PRINT "NOME="NOM$
824        PRINT "END.="EDR$
826         PRINT "CIDADE="CID$
828          PRINT "ESTADO="EST$
830           LOCATE 0,19
832            PRINT "Pressione qquer tecla"
834             K$ = INKEY$:IF K$="" THEN GOTO 834
836              LOCATE 0,19
838               PRINT SPC(30)
840                GOTO 230
1100 ' ------> Mensagens de Auxilio
1110 MSG$ = "Digitar o dia entre 01 e 31"
1112       GOTO 1180
1120 MSG$ = "Digitar o numero do mes entre 01 e 12"
1122       GOTO 1180
1130 MSG$ = "Digitar o ano"
1132       GOTO 1180
1140 MSG$ = "Digitat o nome com ate 25 letras"
1142       GOTO 1180
1150 MSG$ = "Digitar o endereco com ate 30 caracteres"
1152       GOTO 1180
1160 MSG$ = "Nome da ciade com ate 20 caracteres"
1162       GOTO 1180
1170 MSG$ = "Digitar a sigla do estado"
1172       GOTO 1180
1180 LOCATE 0,19
1182  PRINT MSG$
1184   LOCATE 0,20
1186    PRINT "Pressione qquer tecla"
1188     K$ = INKEY$:IF K$="" THEN GOTO 1188
1190      LOCATE 0,19
1192       PRINT SPC(LEN(MSG$))
1194        LOCATE 0,20
1196         PRINT SPC(30)
1198          RETURN
1210 DIA$ = SPACE$(2)
1212        RETURN
1220 MES$ = SPACE$(2)
1222        RETURN
1230 ANO$ = SPACE$(2)
1232        RETURN
1240 NOM$ = SPACE$(25)
1242        RETURN
1250 EDR$ = SPACE$(30)
1252        RETURN
1260 CID$ = SPACE$(20)
1262        RETURN
1270 EST$ = SPACE$(2)
```

```
1272          RETURN
1300 ' ------> Explicacoes
1310 CLS
1312 PRINT "As teclas de controle sao:"
1314 PRINT
1316 PRINT "ctrl-A Apresenta uma orientacao"
1318 PRINT "Ctrl-P Finaliza dados"
1320 PRINT "Ctrl-L Limpa o campo"
1322 PRINT "RETURN Entrada do dado"
1324 PRINT "<< ou BS Retrocede o cursor"
1326 PRINT "--> ou v Avanca o cursor"
1328 PRINT "<-- ou ^ Retrocede o cursor"
1330 PRINT "HOME    Volta o cursor"
1332 PRINT "ESC     Apaga o campo"
1334 PRINT "?       Apresenta esta explicacao"
1336 K$=INKEY$:IF K$="" THEN GOTO 1336
1338 GOTO 400
```

ORIENTAÇÃO AO DIGITADOR

Não deixe o coitado do digitador desamparado. Na medida do possível, faça com que o programa forneça orientações.

Uma mensagem do tipo:

"Pressione qualquer tecla para continuar."

parece, a um programador, totalmente dispensável, porém, existem situações em que o programa não fornecendo nenhuma orientação, provocará dúvidas no usuário quanto ao procedimento a seguir.

Para facilitar a introdução de tais mensagens no programa, sugere-se elaborar uma rotina do tipo apresentado na figura 3.21 .

Figura 3.21 - Rotina "Aguarda Qualquer Tecla".

```
5200 ' ------------------------------
5202 ' QQUER.FNT = Aguarda Qquer Tecla
5204 ' ------------------------------
5205 C=POS(0):M=CSRLIN
5206 PRINT "Pressione qquer tecla para continuar ...";
```

```
5210 TIME = 0
5212 K$ = INKEY$
5214   IF TIME > 200 THEN GOSUB 5030:GOTO 5210
5216   IF K$ = "" THEN GOTO 5212
5217 LOCATE C,M:PRINT SPACE$(40)
5218 RETURN
```

SAVE "QQUER.FNT",A

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

Além desse tipo de orientação ao usuário, o programa poderá emitir sons de tonalidade e duração diferentes para retratar situações distintas.

Existem situações em que o usuário cometeu algum erro de digitação, outras em que o usuário, distraído, deve ser chamado a atenção.

Uma rotina do tipo apresentado na figura 3.22 poderá proporcionar sons adequados para cada uma dessas situações.

Figura 3.22 - Rotina de Sons.

```
5000 ' ---------------------------
5002 ' SONS.FNT = Rotinas de Sons
5004 ' ---------------------------
5010 ' -------> Som de Vai mal
5012           PLAY "L13bL9c"
5014            RETURN
5020 ' -------> Som de Siga
5022           PLAY "L15b"
5024            RETURN
5030 ' -------> Som de Atencao
5032           PLAY "O5V9L9gR9dV8"
5034            RETURN
```

SAVE "SONS.FNT",A

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

ROTINA PARA A ENTRADA DE DADOS

A seguir é apresentada uma rotina genérica para a entrada de dados. Veja a figura 3.23.

Figura 3.23 - Rotina de Entrada de Dados.

```
1000 ' -----------------------------
1002 ' ROTENT.FNT - Rotina de Entrada
1004 ' -----------------------------
1006   A$="":PT=0
1008   LOCATE ,,1
1010 REM
1012 K$=INKEY$:IF K$="" THEN GOTO 1012
1014     IF ASC(K$)= 8 THEN CASO=1:GOTO 1040
1016     IF ASC(K$)=11 THEN CASO=2:GOTO 1040
1018     IF ASC(K$)=12 THEN CASO=3:GOTO 1040
1020     IF ASC(K$)=13 THEN CASO=4:GOTO 1040
1022     IF ASC(K$)=27 THEN CASO=5:GOTO 1040
1024     IF ASC(K$)=28 THEN CASO=6:GOTO 1040
1026     IF ASC(K$)=29 THEN CASO=7:GOTO 1040
1028     IF ASC(K$)<32 OR ASC(K$)>127 THEN A$=K$:
                                           GOTO 1138
1030 A$=A$+K$
1032  PT= PT + 1
1034   PRINT K$;
1036     IF LEN(A$)=CS THEN GOTO 1138
1038      GOTO 1010
1040 IF PT=0 THEN A$=K$:GOTO 1138
1042 ON CASO GOSUB 1048,1066,1078,1098,1106,1118,1128
1044 IF ASC(K$)=13 THEN PRINT CHR$(8);:A$=LEFT$(A$,PT):
                                              GOTO 1138
1046 GOTO 1012
1048 ' ------> 8 = Volta apagando
1050    IF PT < 1 THEN RETURN
1052    IF PT > LEN(A$) THEN GOTO 1138
1054     PT = PT-1
1056      A$=LEFT$(A$,PT)+RIGHT$(A$,LEN(A$)-PT-1)
1058       C=POS(0):M=CSRLIN
1060        PRINT CHR$(8)RIGHT$(A$,LEN(A$)-PT)" ";
1062         LOCATE C,M:PRINT CHR$(8);
1064          RETURN
1066 ' ------> 11 = Cursor no inÀcio
1068               FOR C=1 TO PT
1070                   GOSUB 1128
1072                   NEXT C
1074               PT=0
1076               RETURN
```

```
1078 ' ------> 12 = Apaga o Campo Todo
1080           FOR C=1 TO PT
1082               GOSUB 1128
1084               NEXT C
1086           PT = 0
1088           C=POS(0)
1090           PRINT SPACE$(CS);
1092           LOCATE C,M:PRINT CHR$(8);
1096           RETURN
1098 ' ------> 13 = Sai da rotina
1100          IF LEN(A$)<>0 THEN RETURN
1102           A$=K$
1104           RETURN
1106 ' ------> 27 = Cursor no fim
1108           N=CS - PT
1110           FOR C=1 TO N
1112               GOSUB 1118
1114               NEXT C
1116           RETURN
1118 ' ------> 28 = Avanca o cursor
1120           IF PT=>LEN(A$) THEN RETURN
1122           PRINT CHR$(28);
1124           PT=PT+1
1126           RETURN
1128 ' ------> 29 = Retrocede cursor
1130           IF PT<=0 THEN RETURN
1132           PRINT CHR$(8);
1134           PT = PT - 1
1136           RETURN
1138 LOCATE ,,0:RETURN
```

```
SAVE "ROTENT.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

4

acesso a discos

Este capítulo apresenta os principais conceitos relacionados com a utilização de arquivos em disco magnético, detalha a montagem e a manutenção de arquivos sequenciais e randômicos e descreve algumas técnicas para a manipulação eficiente das informações armazenadas nos arquivos.

O disco magnético, comumente conhecido como "disquete", é composto de trilhas onde são registrados os sinais magnéticos que representam as informações armazenadas no disco pelo programa.

Cada uma das trilhas do disco é dividida em setores, havendo uma determinada quantidade de setores ao longo de uma trilha e essa quantidade de setores é a mesma em todas as trilhas do disco.

Em um setor do disco pode ser armazenada uma quantidade fixa de bytes de informação e essa quantidade é a mesma em todos os setores do disco.

A quantidade de bytes por setor, a quantidade de setores por trilha e a quantidade de trilhas por disco é fixada pelo sistema operacional, em função da tecnologia utilizada na fabricação do disk drive.

Dizer que um determinado disk drive possui uma capacidade maior de armazenamento que outro significa dizer que os bytes são gravados em uma densidade maior e, para não diminuir a qualidade ou a confiabilidade dos dados nele gravados, a tecnologia empregada na fabricação desse disk drive deve ser mais avançada.

O padrão internacional estabelecido para o sistema MSX é o seguinte:

- tamanho em polegadas	= 3,5
- quantidade de faces utilizadas por disco	= 2
- quantidade de trilhas por face	= 80
- quantidade de setores por trilha	= 9
- quantidade de bytes por setor	= 512

Disso resulta que nos disquetes empregados no sistema MSX, podemos armazenar o total máximo de 737.280 bytes, ou seja, 720 Kbytes.

No Brasil, os primeiros drives usados com os MSX foram os mesmos já existentes para o IBM-PC, de 5,25 polegadas e 40 trilhas por face, o que acarreta uma capacidade de armazenamento de 360 Kbytes. Os drives de 3,5 polegadas só apareceram no mercado nacional em 1988.

Não se iluda pensando que todos esses 720 ou 360 Kbytes poderão ser utilizados para o armazenamento de informações, pois nem todo o espaço do disco é disponível para arquivos.

O espaço útil do disco é dividido em 3 áreas distintas, com finalidades específicas. Essas áreas são conhecidas como:

- área do Sistema Operacional
- área do Diretório
- área de Arquivos

A área do Sistema Operacional contém a parte do MSXDOS ou CP/M encarregada de manipular as informações que transitam entre a memória e o disco. Como sabemos, o núcleo principal do Sistema Operacional está gravado na memória ROM e esse núcleo interpreta e executa todos os comandos que são fornecidos ao computador. Conectando-se um disk drive, o computador passa a poder armazenar informações em discos magnéticos, o que é feito mediante a digitação de comandos específicos para essa finalidade. A parte do Sistema Operacional que interpreta e executa esses comandos fica armazenada nas primeiras trilhas do disco e durante a operação de inicialização do sistema, comumente conhecida como "BOOT", ela é transferida para a memória onde é integrada ao núcleo principal do MSXDOS ou CP/M.

A área do Diretório contém os nomes de todos os arquivos gravados no disco e os endereços dos setores utilizados por cada um deles. Na execução do co-

mando FILES, por exemplo, o sistema operacional consulta o diretório para saber quais arquivos estão gravados no disco e o tamanho de cada um deles.

Na área de Arquivos são gravados os arquivos definidos pelos usuários. Cada arquivo do disco tem o seu nome registrado no diretório e as informações do arquivo ficam gravadas em setores de tamanho fixo. Um arquivo que tenha, por exemplo, 700 bytes de tamanho ocupará 2 setores do disco no MSX ou no PC e 6 setores no CP/M.

Toda e qualquer informação manipulada pelos programas pode ser armazenada no disco na forma de arquivos.

Entende-se como "gravação" o ato de se registrar uma certa informação no disco, isto é, transferir a informação que está na memória do computador para o disco magnético onde a mesma poderá ficar armazenada para sempre.

Entende-se como "leitura" o ato de se recuperar uma certa informação gravada anteriormente no disco, isto é, transferir para a memória, uma certa informação que foi "gravada" no disco.

Para podermos gravar e ler informações no disco, devemos seguir certos procedimentos padronizados para garantir que as informações que serão gravadas, o sejam em uma posição "conhecida" para permitir sua posterior leitura. De nada adianta guardar, de qualquer jeito, uma certa informação se não soubermos posteriormente onde ela foi guardada.

Os procedimentos necessários para a gravação de uma informação no disco são os seguintes:

1 - Escolher um nome para o arquivo.

As informações ficam armazenadas no disco na forma de "arquivos". Um disco pode conter uma grande quantidade de arquivos distintos. O sistema operacional distingue um arquivo do outro apenas pelo nome que fica gravado no Diretório do disco.

O nome do arquivo deve ser formado por um conjunto de no máximo 11 letras e algarismos, divididos em duas partes: a primeira, conhecida como "nome do arquivo", é composta de 8 caracteres e a segunda, conhecida como "terminação do nome do arquivo", é composta de 3 caracteres. Entre o "nome" e a "terminação" deve ser colocado um ponto (".") separador.

Exemplos: CADASTRO.ARQ
MOVTO.JAN
FILIAL12.MOV

2 - Executar o comando para abrir o arquivo.

Como em um arquivo convencional, para podermos guardar ou acessar uma informação, devemos, antes de mais nada, abrir a gaveta onde as informações estão arquivadas.

Quando o sistema operacional executa o comando para abrir um arquivo, pode ocorrer uma das duas situações seguintes:

- o arquivo já existe.

Neste caso, as informações serão gravadas "sobre" aquelas existentes, ou seja, as informações anteriormente gravadas são simplesmente apagadas e no lugar das mesmas são gravadas as novas informações.

- o arquivo é novo.

Neste caso, o Diretório é acessado, adicionado o nome desse novo arquivo e reservada uma área nova (sem uso) para a gravação das informações.

3 - Executar o comando para a gravação.

Este comando indica quais informações deverão ser transferidas para o disco e efetiva a gravação.

4 - Executar o comando para fechar o arquivo.

Normalmente, para ganhar tempo, o sistema operacional opera com "buffers" que nada mais são que áreas reservadas, na própria memória para o armazenamento temporário das informações.

O tempo que o sistema operacional gasta para efetivamente gravar uma informação é relativamente grande, pois para conseguir isso, o sistema necessita, entre outras atividades:

- colocar o disco em movimento, pois, quando o disco não está sendo acessado, ele está parado para economizar energia elétrica e diminuir o desgaste mecânico do drive.

- posicionar o cabeçote de leitura e gravação sobre a trilha desejada.

- acionar os circuitos eletrônicos que produzem a gravação magnética.

Suponhamos que um programa deseje gravar apenas a idade de um grupo de pessoas. Ora, a idade de pessoas é uma informação constituida de apenas 2 dígitos e que, portanto, irá consumir apenas 2 bytes no disco para cada pessoa. O programa, naturalmente, irá buscar essas idades em algum outro lugar, no teclado por exemplo, e depois disso, irá gravar no disco, repetindo esse ciclo até completar as idades de todas as pessoas do grupo.

Se o sistema operacional tivesse que "ir ao disco" a cada comando de gravação, o programa perderia um tempo relativamente grande, pois a cada ida ao disco, ele teria que ficar esperando a execução de todo o ciclo de gravação acima apresentado, para a gravação de apenas 2 bytes.

Em vez de "ir ao disco", o sistema operacional "vai ao buffer" e lá armazena os 2 bytes. Na repetição do ciclo, o sistema operacional faz a mesma coisa adicionando os novos 2 bytes aqueles já "gravados". Somente quando o "buffer" já está cheio é que o sistema operacional efetivamente produz a gravação no disco, transferindo não apenas 2 bytes mas todos os bytes que estão no "buffer" de uma só vez.

A transferência do conteúdo do "buffer" para o disco é feita automaticamete pelo sistema operacional quando o buffer fica "cheio", de modo que quando o programa terminar todas as gravações necessárias, pode ocorrer de o buffer não estar totalmente cheio. A execução de um comando de "fechamento" do arquivo, garante que o resto de informações do buffer que ainda não foram gravadas no disco sejam efetivamente transferidas e gravadas no disco.

Os procedimentos necessários para a leitura de informações no disco são os seguintes:

1 - Fornecer ao sistema operacional o nome do arquivo que se deseja acessar.

Ao executar o comando, pode ocorrer uma das duas situações seguintes:

- o arquivo já existe.

Neste caso um dos buffers disponíveis será alocado exclusivamente para o arquivo e os setores onde estão gravadas as informações do arquivo ficarão disponíveis para o programa acessar.

- o arquivo não existe.

Neste caso o sistema operacional apresenta uma mensagem de erro, indicando a inexistência do arquivo.

2 - Executar o comando para a leitura.

Este comando indica quais informações, do arquivo, deverão ser transferidas do disco para a memória.

3 - Executar o comando para "fechar" o arquivo.

Com o "fechamento" do arquivo, o buffer a ele alocado ficará disponível para ser alocado a outro arquivo.

Os arquivos podem ser do tipo Sequencial ou Randômico e os comandos envolvidos na gravação e leitura de informações no disco são os seguintes:

OPEN	**FIELD**	**PRINT #**
INPUT #	**PUT #**	**GET #**
CLOSE		

O comando OPEN indica ao sistema operacional a intenção de se trabalhar com um certo arquivo e na execução do comando devem ser indicados:

- o número do drive onde está inserido o disco que contém ou que deverá conter o arquivo.

- o nome do arquivo.

- o modo de acesso ao arquivo, que poderá ser INPUT, OUTPUT ou APPEND.

- o número de referência do arquivo.

- o tamanho dos registros.

- o tipo de acesso ao arquivo, se sequencial ou randômico.

O comando FIELD aplica-se somente a arquivos do tipo randômico e indica os campos e respectivos tamanhos que constituem um registro.

Os comandos PRINT# e INPUT# são empregados para a gravação e leitura de dados em um arquivo sequencial.

Os comandos PUT# e GET# são empregados para a gravação e leitura de dados em um arquivo randômico.

O comando CLOSE é empregado para encerrar o trabalho com o arquivo. Ele "fecha" o arquivo e libera o buffer correspondente para ser utilizado por um outro arquivo que, porventura, o programa venha a abrir.

Em resumo, os camandos de acesso a arquivos são os seguintes:

	TIPO DE ARQUIVO	
	SEQUENCIAL	RANDôMICO
- Abrir o arquivo	**OPEN**	**OPEN**
- Gravar uma informação	**PRINT #**	**PUT #**
- Ler uma informação	**INPUT #**	**GET #**
- Fechar o arquivo	**CLOSE**	**CLOSE**

ARQUIVO SEQUENCIAL

Os arquivos sequenciais, como o próprio nome indica, são formados por registros gravados um após o outro segundo uma sequência cronológica e possuem as seguintes características:

- os registros podem ter tamanhos diferentes.
- devem ser acessados somente para leitura ou somente para gravação.
- o acesso deve ser sequencial.

Em outras palavras, se o arquivo for aberto para leitura (FOR INPUT) não será possível realizar gravações no mesmo, a não ser que o arquivo seja fechado e, a seguir, aberto para gravação (FOR OUTPUT).

A primeira leitura ou gravação, logo após a abertura do arquivo, será sempre no primeiro registro, a segunda leitura ou gravação será sempre no segundo registro e essa correspondência deve ser mantida até o último registro do arquivo.

Isso significa que caso o programa deseje ler o 48º registro do arquivo, deverá ler, obrigatoriamente, todos os 47 registros que o antecedem.em.

Os comandos que permitem o acesso as informações de um arquivo sequencial são os seguintes:

OPEN   PRINT #   INPUT #   CLOSE

O comando OPEN fornece ao sistema operacional as informações necessárias para preparar o arquivo.

Para as diversas opções de abertura de um arquivo sequencial chamado "CADASTRO.ARQ", que está ou será montado no drive A:, podemos utilizar um dos seguintes comandos:

OPEN "A:CADASTRO.ARQ" FOR OUTPUT AS #1

quando se deseja abrir um arquivo sequencial onde os dados serão gravados pela primeira vez.

O programador deve tomar certos cuidados para evitar que a gravação seja feita "em cima" de um outro arquivo, verificando se já existe outro arquivo com o mesmo nome, através do comando:

FILES "A:CADASTRO.ARQ"

antes de executar o programa, ou embutir no próprio programa a verificação automática da existência de outro arquivo com o mesmo nome. Mais adiante, neste capítulo, apresentaremos algumas sugestões de como isso pode ser feito.

OPEN "A:CADASTRO.ARQ" FOR APPEND AS #1

quando se deseja abrir um arquivo sequencial criado em um outro processamento para adicionar novas informações a ele sem destruir as gravadas anteriormente.

OPEN "A:CADASTRO.ARQ" FOR INPUT AS #1

quando se deseja abrir um arquivo sequencial já existente onde iremos apenas consultar as informações gravadas, sem pretender efetuar qualquer tipo de alteração nas mesmas.

O comando PRINT# indica quais informações devem ser gravadas no arquivo. O seu funcionamento é parecido com o comando PRINT, diferindo apenas pelo parâmetro #n que indica o número de referência do arquivo.

Exemplo: Seja um programa que recebe os seguintes dados pelo teclado:

- nome da pessoa
- apelido
- idade
- número do telefone

e grava o nome, o apelido e a idade em um arquivo chamado "AMIGOS.ARQ" e, o apelido e o número do telefone, em um outro arquivo chamado "TELEFONE.TEL".

Neste caso, o programa irá trabalhar com 2 arquivos simultaneamente, de modo que para cada um deles precisamos ter uma conjunto distinto de OPEN, PRINT# e CLOSE.

A listagem do programa seria algo do tipo:

```
200 MAXFILES = 2
201 PRINT "Inicio do Programa"
203 OPEN "AMIGOS.ARQ" FOR OUTPUT AS #1
204 OPEN "TELEFONE.TEL" FOR OUTPUT AS #2

300 CLS
301 INPUT "Nome ";N$
```

```
302 IF N$="FIM"THEN GOTO 500
303 INPUT "Apelido ",A$
304 INPUT "Idade ";I$
305 INPUT "Telefone ";T$

402 PRINT #1,N$:PRINT #1,A$:PRINT #1,I$
403 PRINT #2,A$,T$

404 GOTO 300

500 PRINT "Final do Programa"
501 CLOSE #1
502 CLOSE #2
503 END
```

Se fosse possível enxergar o que foi gravado em um arquivo sequencial, veríamos algo do tipo:

```
Carlos de Álmeida♪◙Carlao♪◙14♪◙Tereza
do Carmo♪◙Terezinha♪◙16♪◙Milton Tanaka
♪◙Japones♪◙12♪◙Aparecido Simoes Sobrin
ho♪◙Cido♪◙13♪◙Geraldo Prates de Olivei
ra♪◙Gera♪◙15♪◙Augusto de Carvalho♪◙Gug
u♪◙11♪◙Vera Clotilde Meira♪◙Clo♪◙10♪◙N
ivaldete dos Santos♪◙Niva♪◙14♪◙
```

Olhando o exemplo, podemos tirar algumas conclusões:

1 - Em um arquivo sequencial, cada um dos campos fica separado dos outros..

2 - As informações são gravadas conforme foram digitadas.

Caso seja executado o seguinte programa:

```
201 PRINT "Inicio do Programa"
203 OPEN "AMIGOS.ARQ" FOR INPUT AS #1
204 IF EOF(1) THEN GOTO 500
302 INPUT #1,X$,Y$
```

```
303 PRINT X$;"+";Y$
304 GOTO 204
500 CLOSE #1
501 PRINT "Fim do Programa"
503 END
```

será apresentado, no vídeo, o seguinte:

```
Carlos de Almeida+Carlao
14+Tereza do Carmo
Terezinha+16
Milton Tanaka+Japones
12+Aparecido Simoes Sobrinho
Cido+13
Geraldo Prates de Oliveira+Gera
15+Augusto de Carvalho
Gugu+11
Vera Clotilde Meira+Clo
10+Nivaldete dos Santos
Niva+14
Fim do Programa
```

isto é, a cada execução da instrução:

```
302 INPUT#1,X$,Y$
```

as próximas 2 informações do arquivo são atribuídas, respectivamente, às variáveis X$ e Y$, de modo que, compete ao programador saber a ordem exata em que as informações foram gravadas para poder efetuar corretamente a leitura.

Na "fotografia" do arquivo, apresentada na página anterior, os caracteres usados como separadores podem ser visualizados e são os de códigos ASCII 13 (&H0D = Carriage Return) e 10 (&H0A = Line Feed).

Em algumas situações, outros caracteres (vírgula, ponto-e-vírgula e espaço) são usados para separar os campos num arquivo.

O caractere aspas (") pode ser gravado num arquivo com o emprego da função CHR$(34) antes e após a informação. Por exemplo, se quisermos gravar a frase "Hoje é um belo dia" com as aspas num arquivo sequencial previamente aberto com o número 1, devemos comandar:

PRINT #1,CHR$(34)"Hoje é um belo dia"CHR$(34)

Como veremos mais adiante, os arquivos de dados, definidos pelo usuário, são mais versáteis de serem manipulados quando são definidos como sendo do tipo Randômico. Os arquivos Sequenciais são mais apropriados para a comunicação do programa com os periféricos.

```
OPEN "CAS:" FOR INPUT AS #1
OPEN "CAS:" FOR OUTPUT AS #1
OPEN "CRT:" FOR OUTPUT AS #1
OPEN "GRP:" FOR OUTPUT AS #1
OPEN "LPT:" FOR OUTPUT AS #1
```

ARQUIVO RANDÔMICO

Os arquivos randômicos, diferentemente dos arquivos sequenciais, permitem o acesso "randômico" a qualquer um dos registros do arquivo, isto é, os registros podem ser acessados **diretamente em qualquer ordem**.

O termo "randômico" provém do inglês Random que significa "sem uma ordem fixa". Isto significa que os registros podem ser lidos ou gravados em qualquer ordem e um certo registro pode ser acessado tantas vezes quantas se fizerem necessárias no programa.

As características principais de um arquivo randômico são as seguintes:

- o tamanho dos registros deve ser fixo, isto é, todos os registros devem ter o mesmo tamanho.

- o acesso pode ser para leitura ou gravação, não se fazendo essa distinção no comando OPEN.

- os registros podem ser acessados em qualquer ordem, podendo ser acessado em primeiro lugar o registro 56, depois o 12, etc.

Os comandos que permitem o acesso as informações de um arquivo randômico são os seguintes:

OPEN FIELD PUT GET CLOSE

O comando OPEN abre um arquivo em disco. Na execução deste comando, duas situações podem ser encontradas:

A - O arquivo não existe.

Neste caso, o nome do arquivo será adicionado no Diretório do disco e uma área não usada será reservada para o arquivo.

B - O arquivo já existe.

Neste caso, o arquivo existente será aberto.

Diferentemente do arquivo sequencial, em que na abertura devemos indicar se o acesso será somente de leitura ou somente de gravação, o arquivo randômico é aberto para quaisquer tipos de acesso, independentemente do fato do arquivo ser utilizado para leituras, gravações ou atualizacoes, de modo que temos apenas uma única sintaxe para o comando OPEN:

OPEN "A:CADASTRO.ARQ" AS #1 LEN=47

onde indicamos o drive onde se encontra o arquivo, o nome do arquivo, o número de referência e o comprimento dos registros.

No caso de arquivos randômicos é obrigatória a indicação do tamanho do registro, no parâmetro LEN, pois é a existência ou não desse parâmetro que vai indicar ao sistema operacional o tipo de arquivo, se sequencial ou randômico.

Uma vez fixado o comprimento, todos os registros do arquivo, do primeiro até o último, devem ter o mesmo comprimento, nenhum byte a mais ou a menos.

O comando FIELD define as informações que fazem parte do registro e os respectivos tamanhos máximos.

Para a montagem do comando FIELD, o programador deve:

1 - Relacionar as informações que fazem parte do registro.

Esta é uma atividade de planejamento, pois o programador deve escolher somente as informações que serão necessárias.

2 - Escolher um nome conveniente para cada um dos campos.

Os campos que compõem um registro são associados a nomes de variáveis. No caso de arquivos randômicos, tais campos devem ser associados a variáveis do tipo string.

Isso deve ser feito também com informações numéricas, que antes de serem gravadas devem ser convertidas em strings.

3 - Fixar o tamanho máximo que a informação ocupará no disco.

A quantidade de bytes que uma informação ocupa no disco, depende do tipo de informação.

Informações tipo strings ocupam 1 byte para cada caractere da informação.

Informações numéricas ocupam 2, 4 ou 8 bytes conforme o tipo de variável numérica empregada:

TIPO DE VARIÁVEL NUMÉRICA	QUANTIDADE DE BYTES NO DISCO
Inteira	2
Real de Precisão Simples	4
Real de Precisão Dupla	8

Exemplo: Montar o comando FIELD para a definição dos seguintes campos de informações:

- Nome do Funcionário
- Idade
- Quantidade de Horas Trabalhadas
- Salário Anual

Após um estudo minucioso, pois "byte is money", chegamos a conclusão que os seguintes tamanhos safisfazem as necessidades dos campos:

CAMPO	TAMANHO MÁXIMO DA INFORMAÇÃO
Nome do Funcionário	30 caracteres
Idade	2 algarismos
Quantidade de Horas Trabalhadas	3 algarismos
Salário Anual	9 algarismos

Escolhendo um nome apropriado de variável para cada campo:

CAMPO	VARIÁVEL
Nome do Funcionário	NF$
Idade	ID$
Quantidade de Horas Trabalhadas	HT$
Salário Anual	SA$

O comando FIELD será:

```
FIELD #1,30 AS NF$,2 AS ID$,4 AS HT$,8 AS SA$
```

Os comandos LSET e RSET servem para transferir a informação de uma variável comum para a variável de definição do campo dentro do comando FIELD.

As informações tipo string são transferidas diretamente:

```
LSET NF$ = NOME$
```

enquanto que as informações numéricas devem ser transferidas convertendo as mesmas de números para strings usando uma função de conversão de acordo com o tipo de variável numérica.

```
LSET ID$ = MKI$(IDADE)
LSET HT$ = MKS$(HORAS)
LSET SA$ = MKD$(SALARIO)
```

Usando-se essas funções, as informações ocuparão sempre a mesma quantidade de bytes no disco, independentemente do tamanho da informação. Assim, independentemente do fato de a variável SALARIO conter a informação 1.00 ou 678450.00 ela ocupará sempre 8 bytes no disco.

Não existe diferença significativa entre o LSET e o RSET.

O comando PUT efetiva a transferência dos dados da memória para o disco.

Na execução do comando, devemos indicar, por meio do número de referência, em qual dos arquivos desejamos que a gravação seja feita, e dar também o número do registro onde a gravação deverá ser efetuada.

Exemplo: O comando:

PUT #2,423

fará com que as informações definidas no comando FIELD sejam gravadas no 423º registro do arquivo aberto com a referência #2.

O comando GET executa o contrário do comando PUT, ou seja, transfere as informações do disco para a memória.

Exemplo: O comando:

GET #1,47

transferirá o conteúdo do 47º registro do arquivo aberto com a referência #1.

O comando CLOSE finaliza as transferência.

Como vimos no início do capítulo, nem sempre a execução do comando PUT ocasiona a ida do sistema operacional até o disco. Ele "faz de conta" que foi até o disco, mas na verdade, foi só até o "buffer".

Vejamos o mesmo exemplo utilizado para exemplificar o uso de um arquivo sequencial, definindo, desta vez, o arquivo como randômico:

A listagem do programa seria algo do tipo:

```
200 MAXFILES = 2
201 PRINT "Inicio do Programa"
203 OPEN "AMIGOS.ARQ" AS #1 LEN=42
204 FIELD #1,30 AS NP$,10 AS AP$,2 AS ID$
205 OPEN "TELEFONE.TEL" AS #2 LEN=18
206 FIELD #2,10 AS AL$,8 AS TL$

300 N = 0
301 CLS
302 INPUT "Nome ";N$
303 IF N$="FIM" THEN GOTO 500
304 INPUT "Apelido ";A$
```

```
305 INPUT "Idade ";I$
306 INPUT "Telefone ";T$

401 N = N + 1
402 LSET NP$ = N$
403 LSET AP$ = A$
404 LSET ID$ = I$
405 PUT #1,N

406 LSET AL$ = A$
407 LSET TL$ = T$
408 PUT #2,N

409 GOTO 301

500 PRINT "Incluidos ";N;"registros."
501 PRINT "Final do Programa"
502 CLOSE #1
504 CLOSE #2
505 END
```

Se fosse possível enxergar o que foi gravado em um arquivo randômico, veríamos algo do tipo:

```
Carlos de Almeida                    Carlao
  14Tereza do Carmo                      Tere
zinha 16Milton Tanaka
Japones    12Aparecido Simoes Sobrinho
    Cido       13Geraldo Prates de Oliv
eira      Gera        15Augusto de Carvalh
o              Gugu         11Vera Clotilde
Meira               Clo          10Nivaldete
dos Santos               Niva         14
```

onde todas as informações ficam gravadas no comprimento especificado no comando FIELD.

Olhando o exemplo, podemos tirar algumas conclusões:

1 - Em um arquivo randômico, cada um dos campos tem o tamanho especificado no comando FIELD e não há separadores de campos.

2 – As informações são gravadas no tamanho especificado pelo comando FIELD, de modo que quando a informação tiver um comprimento menor, o restante do campo será preenchido com espaços em branco.

Caso seja executado o seguinte programa:

```
201 PRINT "Inicio do Programa"

202 OPEN "ARQUIVOS.ARQ" AS #1 LEN=42
203 FIELD #1,30 AS NF$,10 AS AP$,2 AS ID$
204 MX = LOF(1)/42

301 FOR N = 1 TO MX
302   GET #1,N
303   PRINT NF$;   ID$
304 NEXT N

500 CLOSE #1
501 PRINT "Fim do Programa"
503 END
```

será apresentado, no vídeo, o seguinte:

```
run
Inicio do Programa
Carlos de Almeida                    14
Tereza do Carmo                      16
Milton Tanaka                        12
Aparecido Simoes Sobrinho            13
Geraldo Prates de Oliveira           15
Augusto de Carvalho                  11
Vera Clotilde Meira                  10
Nivaldete dos Santos                 14
Fim do Programa
Ok
```

O arquivo, sendo randômico, pode ser acessado em qualquer ordem, inclusive do fim para o começo.

Caso seja executado o programa:

```
201 PRINT "Inicio do Programa"
202 OPEN "AMIGOS.ARQ" AS #1 LEN=42
203 FIELD #1,30 AS NF$,10 AS AP$,2 AS ID$
204 MX = LOF(1)/42
301 FOR N=MX TO 1 STEP -1
302   GET #1,N
303   PRINT NF$;ID$
304 NEXT N
500 CLOSE #1
501 PRINT "Fim do Programa"
502 END
```

será apresentado, no vídeo, o seguinte:

```
run
Inicio do Programa
Nivaldete dos Santos              14
Vera Clotilde Meira               10
Augusto de Carvalho               11
Geraldo Prates de Oliveira        15
Aparecido Simoes Sobrinho         13
Milton Tanaka                     12
Tereza do Carmo                   16
Carlos de Almeida                 14
Fim do Programa
Ok
```

ACESSO AO DISCO

Os arquivos sequenciais e randômicos são arquivos criados e mantidos pelos programas desenvolvidos pelo usuário.

Além desses tipos, existem os arquivos criados e mantidos pelo sistema operacional. São os arquivos de programas.

Esses arquivos, parecidos com o arquivo sequencial, são constituidos de registros de tamanhos variados, contendo, em cada registro, o seguinte:

- número da instrução em 2 bytes
- código "token" do comando em 1 byte
- parte de texto da instrução
- separadores 0D0A entre uma instrução e outra
- finalizador 1A para indicar o fim do arquivo.

Obviamente, isso é válido para os arquivos de programas em BASIC. Programas feitos em outras linguagens são armazenados de forma diversa.

A manipulação desses arquivos, isto é, a transferência entre a memória e o disco, é efetuada com o auxílio dos comandos seguintes:

LOAD **SAVE**

O comando LOAD efetua a transferência de um programa do disco para a memória.

Para a execução do comando, devemos especificar em qual dos drives está o programa, ou havendo o mesmo programa em vários drives, qual deles desejamos carregar e o nome do programa. Exemplo:

LOAD "A:CADASTRA"

Na execução do comando, poderão ocorrer 2 situações diferentes:

- o programa encontrado no disco está gravado na forma de "token".

  Neste caso, a transferência será direta.

- o programa encontrado no disco está gravado na forma de texto.

Neste caso, a transferência do programa para a memória será efetuada com a conversão dos comandos para o "token" correspondente.

Maiores detalhes sobre a forma de armazenamento de programas na memória podem ser obtidos na página 23 do livro "Aprofundando-se no MSX".

O comando SAVE efetua a transferência do programa da memória para o disco, gravando o programa em um arquivo.

Para a execução do comando, devemos especificar em qual dos drives desejamos gravar o programa e o nome do mesmo. Exemplo:

SAVE "B:CADASTRA.BAS"

Na execução do comando, duas situações distintas poderão ocorrer:

- O arquivo não existe.

Neste caso, o sistema operacional inclue o nome do programa no diretório do disco e efetua a gravação do programa na forma de um arquivo com o nome CADASTRA.BAS.

- O arquivo já existe.

Se já existir no disco um arquivo com o nome CADASTRA.BAS, independentemente do fato de ser ou não um arquivo de programa, podendo inclusive ser um arquivo de dados, o sistema operacional simplesmente libera a área ocupada pelo arquivo existente e grava, em cima, o programa que está na memória.

A gravação do programa resume-se em efetuar uma cópia do programa que está na memória, de modo que os comandos são gravados na forma de "token" a exemplo daquilo que está na memória.

O programador poderá querer que o programa seja gravado no disco na forma de texto, com os comandos gravados conforme digitados, para, por exemplo, ser editado com o auxílio de um Processador de Textos. Neste caso, basta adicionar o parâmetro "A" para indicar ao sistema operacional que todos os "tokens" devem ser convertidos nas palavras de comando correspondentes. Exemplo:

SAVE "B:CADASTRA.BAS",A

Os comandos LOAD e SAVE podem ser usados dentro de programas. Existem situações em que um determinado programa não cabe por inteiro na memória e situações em que mesmo cabendo, não nos interessa carregá-lo por inteiro, pois diminuiria consideravelmente a área livre na memória, causando algumas restrições quanto a quantidade de buffers ou a quantidade de variáveis definíveis na memória.

Nessas situações, o correto é dividir o programa em 2 ou mais partes e fazer com que uma parte carregue, automaticamente, a outra parte para prosseguir com o processamento.

Uma das maneiras de fazer com que uma parte do programa carregue a outra parte, a partir do disco, é utilizar o comando RUN, mas este comando, tem a propriedade de fechar todos os arquivos abertos.

O comando LOAD com o parâmetro "R" oferece uma saída elegante, não sendo necessário, no programa que está sendo carregado, abrir novamente os arquivos em uso, isto é, todos os arquivos abertos pelo primeiro programa continuarão abertos após a carga do segundo programa.

Vejamos, a seguir, algumas técnicas de acesso ao disco objetivando o seguinte:

- facilitar e aumentar a velocidade de desenvolvimento de um programa

- facilitar e diminuir o tempo gasto na interpretação da listagem de um programa.

1 - Padronizar e agrupar os comandos de abertura de arquivos.

Na digitação dos comandos de abertura de arquivos pode ocorrer um erro de digitação que possa por a perder todo o funcionamento do programa.

Recomenda-se procurar digitar o menos possível para evitar erros de digitação.

Um grande sistema é geralmente composto por vários programas que manipulam diversos arquivos. Um desses arquivos poderia, por exemplo, apresentar as seguintes instruções de abertura:

```
200 OPEN "B:MOVI1.IPN" AS #1 LEN=86
201 FIELD #1,20 AS N$,15 AS N1$,10 AS A$$,5 AS B5$
```

Veja bem, são instruções de digitação difícil, podendo-se cometer facilmente algum erro de digitação. Se esse arquivo é um daqueles que é manipulado em todos os programas, essas instruções deverão ser digitadas em todos eles. Fatalmente algum erro de digitação será cometido.

Podemos criar um arquivo contendo todos os gabaritos de arquivos utilizados pelos programas do sistema. Tal arquivo pode ser gravado na forma de tex-

to com o nome de GABARITO.FNT e quando um novo programa precisar de um dos gabaritos, poderíamos carregar o gabarito que nos interessa com a instrução:

```
MERGE "GABARITO.FNT"
```

Exemplo: Vamos supor que os programas PROG1, PROG2 e PROG3 acessam o mesmo arquivo, de modo que as instruções:

```
OPEN "A:CADASTRO.ARQ" AS #1 LEN = 88
FIELD #1,30 AS N$,50 AS E$,8 AS S$
```

devem estar presente nos 3 programas.

Afim de evitar erros de digitação por exemplo, queremos evitar de digitar essas instruções em cada um desses programas. Para isso:

1 - Digitar somente as instruções comuns e salvar no disco na forma de texto.

```
NEW
200 OPEN "A:CADASTRA.ARQ" AS #1 LEN=88
202 FIELD #1,30 AS N$,50 AS E$,8 AS S$

SAVE "GABARITO.FNT",A
```

2 - Incluir as instruções em cada um dos 3 programas:

```
LOAD  "PROG1"
MERGE "GABARITO.FNT"
SAVE  "PROG1"

LOAD  "PROG2"
MERGE "GABARITO.FNT"
SAVE  "PROG2"

LOAD  "PROG3"
MERGE "GABARITO.FNT"
SAVE  "PROG3"
```

2 - A numeração das linhas dos comandos de abertura dos arquivos poderá ser padronizada entre 6000 e 6999 e a execução da abertura poderá ser efe-

tuada por um comando GOSUB. Assim, todos os comandos de abertura de todos os arquivos que possam ser necessários aos diversos programas do sistema ficarão agrupados entre as linhas 6000 e 6999.

Isto fará com que a listagem do programa seja menos poluída, pois no núcleo principal teremos apenas uma instrução do tipo:

```
200 GOSUB 6010
```

sem a poluição causada pelas instruções:

```
200 OPEN "A:CADASTRA.ARQ" AS #1 LEN = 88
202 FIELD #1,30 AS N$,50 AS E$,8 AS S$
```

Neste caso, a rotina fonte deve ser digitada conforme o seguinte:

```
6010 OPEN "A:CADASTRA.ARQ" AS #1 LEN = 88
6012 FIELD #1.30 AS N$,50 AS E$,8 AS S$
6014 RETURN


SAVE "GABARITO.FNT",A
```

3 - O número do drive pode ser definido em uma variável, para que o programador possa, rapidamente, definir os arquivos em drives distintos, o que é importante, principalmente na fase de implantação do sistema.

Em vez de:

```
OPEN "A:CADASTRO" AS #1 LEN=47
```

Utilizar:

```
OPEN DC$+"CADASTRO" AS #1 LEN=47
```

onde a variável DC$ possa ser definida no início do programa com:

```
DC$ = "A:"
```

ou lida, apartir do teclado com:

```
INPUT "Em qual drive esta o Cadastro ";DC$
```

ou, ainda, lida a partir de um arquivo de parâmetros

onde está gravada a configuração do sistema de arquivos.

Essa possibilidade é muito importante nos casos em que o programador pretenda fazer farta distribuição do programa, pois haverá interessados que dispõem de apenas 1 único drive, enquanto que outros, mais afortunados, disporão de muitos drives. Nestes casos, o programador não precisará percorrer a listagem, instrução por instrução, a procura dos A: e B: para serem alterados.

4 - As mesmas observações acima podem ser aplicadas ao nome do arquivo. Isto é muito importante nos casos de programas que executam processamentos mensais.

Nestes casos, é comum utilizarmos a identificação do mês como parte do nome dos arquivos, como por exemplo:

MOVTO.JAN
PARAM.JAN

Definindo-se o nome do arquivo em uma variável, o mesmo programa poderá ser executado com arquivos de meses diferentes:

```
100 INPUT "Qual o mes ";ME$
101 ARQ$ = "MOVTO."+ME$
102 OPEN "A:"+ARQ$ AS #1 LEN=47
```

5 - Os arquivos acessados pelo programa não necessitam estar todos simultaneamente abertos. Em cada parte do programa, devemos abrir somente aqueles arquivos que serão acessados por aquela parte e tão logo um determinado arquivo não seja mais necessário, devemos proceder ao seu fechamento, inclusive para maior proteção e segurança dos dados.

Desse modo, se um programa for operar com 7 arquivos, não é racional abrir todos eles no início do programa e fechá-los, todos, somente ao final do programa. A não ser que isso seja realmente necessário.

Lembre-se que a quantidade máxima normal de arquivos que podem estar simultaneamente abertos é fixada pelo sistema operacional durante a inicialização do sistema, quando são definidos os "buffers", um para cada arquivo. Havendo a necessidade de manipulação de

mais de um arquivo simultaneamente, o arquivo deverá definir mais espaço para os "buffers". No MSX, por exemplo, isso pode ser feito pelo comando MAXFILES.

Lembre-se também que existe uma correspondência inversa entre a quantidade de "buffers" e o espaço disponível na memória para o programa.

Para poder operar somente com aqueles arquivos necessários em cada parte do programa, o número de referência do arquivo pode ser definido em uma variável, como no exemplo seguinte:

```
OPEN "A:CADASTRO" AS # NARQ LEN=47
```

6 - Para facilitar a compreensão da estrutura dos diversos arquivos acessados pelo programa, as instruções de abertura e correspondentes instruções de definição de campos podem ser agrupadas em um certo trecho do programa, sendo executados por meio de comandos GOSUB.

```
6000 ' -----------------------------------
6002 ' ABREARQ.FNT = Abertura de Arquivos
6004 '-----------------------------------
6010 ' ---------> Arquivo de Parametros
6012 OPEN DR$+"CMPAR.ARQ" AS #NARQ LEN=51
6014 FIELD #NARQ,30 AS TT$,2 AS NM$,9 AS DM$,2 AS RC$,
                 2 AS RM$,2 AS UD$,2 AS UM$,2 AS UA$
6016 RETURN
6020 ' ------> Cadastro de Materiais
6022 OPEN DR$+"CMCAD.ARQ" AS #NARQ LEN=63
6024 FIELD #NARQ,4 AS CD$,25 AS DC$,10 AS TC$,4 AS SR$,
                 4 AS SL$,4 AS QM$,6 AS UE$,6 AS US$
6026 RETURN
6030 ' ------> Arquivo de Movimento
6032 OPEN DR$+"CMMOV.ARQ" AS #NARQ LEN=25
6034 FIELD #NARQ,6 AS MD$,10 AS MH$,4 AS MC$,4 AS MQ$,
                 1 AS MV$
6036 RETURN
```

```
SAVE "ABREARQ.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

7 - O comando FIELD, como vimos, define a estrutura do arquivo randômico, separando os campos que constituem o registro.

De acordo com o objetivo de um trecho do programa, os campos contíguos podem ser agrupados em outros campos para facilitar a manipulação dos mesmos.

Nestes casos, podemos encadear vários comandos FIELD, contendo em cada um, a estrutura mais conveniente para a manipulação adequada das informações.

```
100 OPEN "CLIENTES.ARQ" AS #1 LEN=77
102 FIELD #1,20 AS NOME$,30 AS EDR$,5 AS CEP$,
             20 AS MUN$,2 AS FED$
104 FIELD #1,50 AS X$,27 AS LCAL$
106 MAX = LOF(1)/77
108 FOR REG = 1 TO MAX
110     GET #1,REG
112     PRINT "Nome = "NOME$
114     PRINT "Endereço = "EDR$
116     PRINT "Localidade = LCAL$
118 NEXT REG
120 END
```

Observe que no programa acima, a instrução:

```
116     PRINT "Localidade = LCAL$
```

equivale a instrução:

```
116     PRINT "Localidade = "CEP$;MUN$;FED$
```

Para compreender bem o funcionamento dos comandos de acesso ao disco, recomenda-se elaborar diversos programas curtos, enfocando, em cada um deles, um aspecto diferente do acesso ao disco.

Para sentir a performance dos acessos, o programador deve executar os testes em cima de arquivos com uma quantidade considerável de registros e, para evitar uma grande perda de tempo digitando um grande arquivo, recomenda-se desenvolver um programa específico que "cria" automaticamente tais arquivos.

```
100 ' ===============================================
102 ' CRIARQ = Cria um arquivo com dados ficticios
104 ' ===============================================
106 ' Roberto Watanabe 22/03/87
108 '
200 , --------------------
202 , Definicoes iniciais
204 , --------------------
210 NARQ=1
```

```
212 DRIVE$="A:"
214 ARQ$="CADASTRO.ARQ"
216 GOSUB 6010
300 ' ------------------------------------
302 ' Cria as informacoes aleatoriamente
304 ' ------------------------------------
310 FOR REG = 1 TO 1000
312   NP$ = ""
314   I% = RND(1) * 20
316   I% = I% + 10
318   FOR J = 10 TO I%
320     NP$ = NP$ + CHR$(65+RND(1)*26)
322   NEXT J
324   SALARIO = RND(1) * 30000
326   LSET NF$ = NP$
328   LSET SF$ = MKS$(SALARIO)
330   PUT #NARQ,REG
332 NEXT REG
400 '-------------------
402 ' Final do Programa
404 '-------------------
410 CLOSE #NARQ
412 PRINT "Final do Processamento"
414 END
6000 '---------------------------------
6002 ' Rotinas de abertura de arquivos
6004 '---------------------------------
6010 ' Cadastru de Funcionarios
6012 OPEN DRIVE$+ARQ$ AS #NARQ LEN=34
6014 FIELD #NARQ,30 AS NF$,4 AS SF$
6016 RETURN
```

ORDENAÇÃO DE ARQUIVOS

Uma das grandes vantagens do computador é sem sombra de dúvida, a sua capacidade de manipular grandes quantidades de dados em pouco tempo.

Dentre as manipulações possíveis, aquela que mais caracteriza a facilidade de tratamento de dados é a possibilidade de ordená-los segundo um certo critério. Na contabilidade, por exemplo, independentemente da ordem em que os dados foram inseridos, deve-se emitir determinados relatórios com os dados ordenados segundo a data, como é o caso do Diário, e outros relatórios devem ser emitidos com os dados ordenados segundo o código das contas, como é o caso do Razão.

No Controle de Estoques, existem relatórios que devem apresentar os materiais por ordem alfabética e outros por ordem de código do produto.

Analisando as muitas aplicações de computadores iremos constatar que em todas elas, havendo a possibilidade de apresentação dos dados ordenados conforme critérios diversos, o programa ganha novos impulsos sendo mais valorizado.

Para o computador é indiferente se, para a realização de um determinado serviço, ele terá que executar algumas poucas ou milhares de intruções.

Compete ao programador, programar o computador para tal.

Manipular dados é sempre uma atividade trabalhosa, repetitiva e cansativa.

Pegue, por exemplo, um relatório de 200 páginas e tire 10 cópias. Isso vai resultar em uma pilha com 2.000 folhas. Agora separe os 10 exemplares. Trabalhoso e cansativo não?

Analisando as diversas alternativas em que isso pode ser feito, iremos verificar que algumas delas, mais racionais, irão ocasionar um trabalho menor.

No caso em questão, se dispusermos de uma grande mesa podemos empregar a técnica de "colocação", isto é, da pilha principal, pegamos um pequeno maço de folhas, e andando ao redor da mesa, vamos "colocando", folha por folha, cada uma das 10 cópias. Completada a colocação das 10 cópias, o maço que está em nossas mãos estará automaticamente posicionado na segunda página do relatório, de modo que podemos repetir a operação anterior, "colocando", desta vez, a segunda página. Repetindo-se esse ciclo 200 vezes, teremos os 10 exemplares perfeitamente separados. Isso, é claro, se a máquina copiadora não cometeu nenhum engano, tirando algumas cópias a mais ou a menos.

A técnica com que iremos manipular os dados, pode depender do tipo de dados que dispomos.

No exemplo de separar as cópias de um relatório, podemos ter um outro caso em que desejamos tirar 200 cópias de um relatório de apenas 10 páginas.

Empregar a técnica anterior neste caso, significa que teremos que ter a nossa disposição, uma enorme mesa onde possamos dispor todas as 200 cópias. Nesse caso, recomenda-se a técnica da "remoção". Nessa técnica, retiramos da pilha principal, onde há 2.000 folhas, maços de exatamente 200 folhas e dispomos tais maços, lado a lado, ao longo da mesa. Teremos, portanto, 10 pilhas distintas. Feito isso, andamos em redor

da mesa, "retirando" uma folha de cada pilha. Após a última pilha, teremos, nas mãos, um relatório completo. Repetindo esse ciclo 200 vezes, teremos separado os 200 exemplares do relatório.

Para a ordenação de dados através de um computador, também não existe apenas uma única técnica.

Muitos pesquisadores elaboraram técnicas especiais e algumas delas ficaram tão famosas que são conhecidas pelo nome do seu autor.

O programador deve analisar alguns aspectos importantes antes de adotar alguma técnica de ordenação de dados. Vejamos algums desses aspectos:

A - Como os dados se encontram e que tipo de classificação é desejada.

B - Qual o tamanho do espaço de trabalho disponível, em outras palavras, qual o tamanho da "mesa"?

Vamos analisar o problema da ordenação de dados, em função da situação em que os dados se encontram, antes da ordenação.

ORDENANDO UM ARQUIVO COM POUCOS REGISTROS FORA DE ORDEM

Vamos supor que um arquivo já se encontra ordenado.

Um cadastro de clientes, por exemplo, encontra-se normalmente ordenado por ordem alfabética de nomes, e nesse cadastro queremos inserir um novo cliente.

Inserir um novo cliente, significa adicionar apenas um novo registro, e neste caso, querer aplicar um algoritmo de ordenação em todo o arquivo, é uma tentativa meio irracional, mesmo que o tal algoritmo seja dos mais eficientes possíveis.

Mesmo se tratando de uma simples inserção, isso não pode ser efetuado diretamente, pois um arquivo em disco não é como um fichário com fichas de papelão, onde inserir uma nova ficha significa "encaixar" a mesma no meio das outras e, se ficar apertado, dar uma empurradinha nas outras fichas.

Nos arquivos em meio magnético, os registros ficam fisicamente "presos" naquelas posições onde foram gravados e a atividade de "deslocar" os registros

é inviável. O que é possível, é a leitura num lugar e a gravação em outro local um pouco deslocado, para fazer de conta que estamos deslocando.

Dependendo do espaço disponível no disco, podemos empregar uma das duas técnicas apresentadas a seguir:

A - Empregar 2 arquivos.

Criar um arquivo temporário e transferir os registros, anteriores àquele que desejamos inserir, para o novo arquivo. Gravar no arquivo novo, o novo registro.

Transferir os demais registros para o arquivo novo.

Exemplo:

```
300 '----------------
302 'Entrada de Dados
304 '----------------
310 INPUT "nome";N$
312 INPUT "endereço";E$
400 '----------------
402 'Abre os Arquivos
404 '----------------
410 OPEN "A:CADASTRO.ARQ" AS #1 LEN=80
412   FIELD #1,30 AS N1$,50 AS E1$
414   FIELD #1,80 AS C1$
416 OPEN "A:CADASTRO.$$$" AS #2 LEN=80
418   FIELD #2,30 AS N2$,50 AS E2$
420   FIELD #2,80 AS C2$
500 '---------------------
502 'Transferencia Inicial
504 '---------------------
510 CONTROLE% = 1
512 MAXREG% = LOF(1) / 80
514 FOR REG%=1 TO MAXREG%
516   GET #1,REG%
518   IF CONTROLE%=2 THEN GOTO 119
520   IF N1$>N$ THEN LSET C2$ = C1$:
                     LSET N2$ = N$ :
                     LSET E2$ = E$ :
                     PUT #32,REG%  :
                     CONTROLE% = 2 :
                     GOTO 610
```

```
522   LSET C2$ = C1$
524   PUT #2,REG%
526   GOTO 614
600 '--------------------
602 'Transferencia Final
604 '--------------------
610   LSET C2$ = C1$
612   PUT #2,REG%+1
614 NEXT REG%
700 '------------------
702 'Final do Programa
704 '------------------
710 MAXREG% = MAXREG%+1
712 PRINT "O Cadastro contem "MAXREG%" registros"
714 CLOSE #1
716 CLOSE #2
718 KILL "A:CADASTRO.ARQ"
720 NAME "A:CADASTRO.$$$" AS "A:CADASTRO.ARQ"
722 END
```

B - Um único arquivo.

Neste caso, iremos "deslocar" os registros, lendo-os e gravando-os um pouco deslocados.

Na alternativa de não se conhecer, a priori, a posição onde o novo registro deve ser encaixado, o programa deve procurar essa posição, varrendo todo o arquivo.

```
300 '-----------------
302 'Entrada de Dados
304 '-----------------
310 INPUT "nome";N$
312 INPUT "endereço";E$
400 '--------------
402 'Abre arquivos
404 '--------------
410 OPEN "A:CADASTRO.ARQ" AS #1 LEN=80
412   FIELD #1,30 AS N1$,50 AS E1$
414   FIELD #1,80 AS C1$
500 '----------------------
502 'Transferencia Inicial
504 '----------------------
510 CONTROLE% = 1
```

```
512 MAXREG% = LOF(1)/80
514 FOR REG%=MAXREG$ TO 1 STEP -1
516   GET #1,REG%
518   IF N1$>N$ THEN PUT #1,REG%+1 : GOTO 528
520   LSET N1$ = N$
522   LSET E1$ = E$
524   PUT #1,REG%+1
526   REG% = 1
528 NEXT REG%
600 '------------------
602 'Final do Programa
604 '------------------
610 MAXREG% = MAXREG% + 1
612 PRINT"O cadastro contem"MAXREG%"regisros"
614 CLOSE #1
616 CLOSE #2
618 END
```

O método B apresenta a vantagem de necessitar de um espaço menor no disco, porém existe a possibilidade de ocorrer um imprevisto durante o deslocameto dos registro, provocando a perda do arquivo.

Se o programa já souber a posição em que o novo registro deve ser encaixado, o programa pode valer-se de uma rotina, como a abaixo apresentada, para deslocar os registros finais e "abrir" uma vaga para poder-se inserir o novo registro.

```
1300 ' ------------------------------------
1302 ' ROTINS.FNT = Insere Novo Material
1304 ' ------------------------------------
1310 IF REG% > MX% THEN GOTO 1358
1312 LOCATE 0,12
1314 PRINT TAB(9)STRING$(22,"*")
1316 PRINT TAB(9)"Estou providenciando a"
1318 PRINT TAB(9)"insercao do registro."
1320 PRINT TAB(12)"NAO INTERROMPA !"
1322 PRINT TAB(9)STRING$(22,"*")
1324 GOSUB 5350
1332 REF = MX% - REG%
1340   FOR C%=MX% TO FIRST% STEP -1
1342   GET #NARQ,C%
1344   PUT #NARQ,C%+1
1346   LOCATE 5+30*(MX%-C%+1)/REF,19
1348   PRINT ">"
1350   NEXT C%
```

```
1352 LOCATE 0,12
1354 PRINT SPC(160) : PRINT SPC(160)
1358 RETURN          +30*(MX%-C%+1)/REF.19
1348       PRINT ">"
5350 ' ----------------------------------------
5352 ' MAPAACC.FNT = Mapa de Acompanhamento
5354 ' ----------------------------------------
5356 'Mostra o estágio do processo
5362 LOCATE 5,17 : PRINT "0%"SPACE$(13)"50%";
5366 PRINT SPACE$(11)"100%" : LOCATE 5,18
5368 PRINT "!";:FOR N=1 TO S:PRINT "    !";:NEXT N
5370 RETURN

      SAVE "ROTINS.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

As técnicas apresentadas são válidas quando se pretende inserir um novo registro em um arquivo já existente.

Vejamos como proceder para colocar em ordem arquivos que possuem alguns registros já gravados fora da ordem, ainda sem apelarmos para um algoritmo de ordenação.

Existem várias técnicas para isso, mas vamos detalhar apenas uma delas, passo a passo.

1 - Varrer o arquivo todo a procura dos registros que estão fora da ordem.

A cada registro que estiver fora da ordem, copiá-lo em um arquivo provisório.

Colocar uma marca, por exemplo "$$$", no registro que foi copiado.

O arquivo provisório ficará, assim, com todos os registros fora de ordem.

2 - Para cada registro do arquivo provisório, varrer o arquivo a procura da posição correta de encaixe do registro.

3 - Varrer novamente o arquivo a procura de um registro livre, que foi marcado com "$$$".

4 - Deslocar os registros entre a posição livre e a posição de encaixe, transferindo a posição livre para a de encaixe.

5 - Copiar o registro do arquivo provisório na posição livre.

6 - Repetir o ciclo a partir do passo 2.

A listagem da rotina é apresentada a seguir:

```
2100 ' ----------------------------------------
2102 ' RTPOEORD.FNT = Poe em ordem sem ordenar
2104 ' ----------------------------------------
2110 GET #A1,1
2112 A$ = N$
2114 MT = 0
2118 ' Retira os registros fora de ordem, jogando-os
       no arquivo provisorio
2120 FOR REG = 2 TO MREG
2122   GET #A1,REG
2124   IF N$ > A$ THEN A$ = N$: GOTO 2140
2126   MT = MT + 1
2128   GET #A1,REG - 1
2129   LSET M$ = N$
2130   PUT #A2,MT
2132   LSET N$ = "$$$"
2134   PUT #A1,REG - 1
2136   GET #A1,REG
2138   A$ = N$
2140 NEXT REG
2150 ' Procura a posicao de encaixe, desloca e encaixa.
2152 FOR RT = 1 TO MT
2154   GET #A2,RT
2156   A$ = M$
2160   GOSUB 2200:'Procura a posicao de encaixe.
2162   IF EX = 0 THEN EX = MREG
2170   GOSUB 2300:'Desloca e encaixa.
2172   GET #A2,RT
2173   LSET N$ = M$
2174   PUT #A1,EX
2176 NEXT RT
2180 RETURN
```

SAVE "RTPOEORD.FNT",A

```
2200 ' ------------------------------------------
2202 ' RTPROCPE.FNT = Procura a posicao de encaixe
2204 ' ------------------------------------------
2210 EX = 0
2212 FOR REG = 1 TO MREG
2214   GET #A1,REG
2216   IF N$ = "$$$" THEN GOTO 2224
2218   IF N$ < A$ THEN GOTO 2224
2220   EX = REG - 1
2222   REG = MREG
2224 NEXT REG
2226 RETURN
```

```
SAVE "RTPROCPE.FNT",A
```

```
2300 ' ------------------------------
2302 ' RTDESENC.FNT = Desloca e encaixa
2304 ' ------------------------------
2308 FOR REG = 1 TO MREG
2310   GET #A1,REG
2312   IF N$ <> "$$$"+SPACE$(17) THEN GOTO 2318
2314   LIV = REG
2316   REG = MREG
2318 NEXT REG
2330 IF LIV < EX THEN PS = 1
2332 IF LIV > EX THEN PS = -1
2334 FOR K = LIV TO EX-PS STEP PS
2336   GET #A1,K+PS
2338   PUT #A1,K
2340 NEXT K
2342 RETURN
```

```
SAVE "RTDESENC.FNT",A
```

Testar a rotina com o programa:

```
100 ' ------------------------------------------
102 ' EXCOSORD.BAS = Coloca em ordem, sem ordenar
104 ' ------------------------------------------
110 ' R.M.Watanabe                       23/05/87
112 '
200 ' ------------------
202 ' Definicoes iniciais
204 ' ------------------
```

```
206   OPEN "cadastro.arq" AS #1 LEN=20
208        FIELD #1,20 AS N$
210   OPEN "arquivo.prv" AS #2 LEN=20
212        FIELD #2,20 AS M$
214 LSET N$ = "Antonio   ":PUT 1,1
216 LSET N$ = "Benedito  ":PUT 1,2
218 LSET N$ = "Carlos    ":PUT 1,3
220 LSET N$ = "Ivo **    ":PUT 1,4
222 LSET N$ = "Denise    ":PUT 1,5
224 LSET N$ = "Edson     ":PUT 1,6
226 LSET N$ = "Felicio   ":PUT 1,7
228 LSET N$ = "Gastao    ":PUT 1,8
230 LSET N$ = "Knese **  ":PUT 1,9
232 LSET N$ = "Helio     ":PUT 1,10
234 LSET N$ = "Jaime     ":PUT 1,11
236 LSET N$ = "Luciana   ":PUT 1,12
238   MREG = 12
240   A1 = 1
242   A2 = 2
300 ' ------------------------------
302 ' Teste da Rotina de Ordenacao
304 ' ------------------------------
310   PRINT "*** INICIAL ***"
312   FOR N = 1 TO MREG
314       GET 1,N
316       PRINT N,N$
318       NEXT N
320   GOSUB 2100
330   PRINT "*** ORDENADO ***"
332   FOR N = 1 TO MREG
334       GET 1,N
336       PRINT N,N$
338       NEXT N
340   PRINT "*** FIM ***"
342    CLOSE #1,#2
344     KILL "CADASTRO.ARQ"
346      KILL "ARQUIVO.PRV"
348       END
```

```
MERGE "RTPOEORD.FNT"
MERGE "RTPROCPE.FNT"
MERGE "RTDESENC.FNT"
SAVE  "EXCOSORD.BAS"
RUN
```

ORDENANDO UM ARQUIVO COM MUITOS REGISTROS FORA DE ORDEM

A ordenação de um arquivo em situações em que a ordem dos registros é desconhecida, isto é, os registros poderão estar totalmente misturados, parcialmente ordenados ou mesmo ordenados segundo uma ordem desconhecida, é uma tarefa um pouco mais complexa, comparada aquela situação em que é conhecido o critério de ordenação.

Nestes casos, antes de adotar um algoritmo eficiente, devemos analisar alguns pontos importantes que influem na velocidade da ordenação.

O algoritmo, qualquer que seja ele, irá efetuar uma certa quantidade de comparações e trocas de posições.

O programador deve estimar o tempo total dessas comparações e trocas. Considerando que tais tempos dependem fundamentalmente da configuração do seu computador, realize alguns ensaios comparativos e anote no seu Caderno de Dicas o tempo unitário, executando programas do tipo:

```
100 TIME = 0
102 FOR N=1 TO 1000
104 NEXT N
106 PRINT "TEMPO= ";TIME/60;"s"
108 END
```

```
100 TIME = 0
102 FOR N=1 TO 1000
104 A = B
106 NEXT N
108 PRINT "TEMPO= ";TIME/60;"s"
110 END
```

```
100 TIME = 0
102 FOR N=1 TO 1000
104 IF A=B THEN GOTO 106
106 NEXT N
108 PRINT "TEMPO= ";TIME/60;"s"
110 END
```

No computador em que efetuei esses ensaios, obtive os seguintes tempos:

- cada loop FOR ... NEXT = 1,96 mseg
- cada transferência do tipo A=B = 1.42 "
- cada comparação com IF = 2,55 "

Suponhamos que desejamos ordenar um certo conjunto contendo 1000 números e o algoritmo empregado efetue N*N/2 comparações e N*N/4 trocas. Neste caso, o tempo total será calculado conforme o seguinte:

- FOR ... NEXT	1000*1000/2 * 1,96 =	960.000
- Comparações	1000*1000/2 * 2,55 =	1.275.000
- Trocas	1000*1000/4 * 1,42 =	355.000

TOTAL = 2.590.000 milissegundos

o que representa cerca de 43 minutos.

Observe que esse tempo, corresponde a ordenação do conjunto, supondo que o mesmo já esteja na memória. No caso de ordenação de arquivos em disco, devemos considerar o tempo gasto para a transferência das informações do disco para a memória e, após a ordenação, o tempo gasto para a transferência das informações da memória para o disco.

O tempo de leitura e de gravação no disco deve ser levantado por meio de ensaios, pois depende também da configuração do seu computador. Utilize um programa do tipo:

```
100 OPEN "A:TESTE.ARQ"AS #1 LEN=1
102 FIELD #1,1 AS CAMPO$
104 LSET CAMPO$ = "A"
106 TIME = 0
108 FOR REG=1 TO 1000
110   PUT #1,REG
112 NEXT REG
114 T1 = TIME : TIME = 0
116 FOR REG=1 TO 1000
118   GET #1,REG
120 NEXT REG
```

```
122 T2 = TIME
124 PRINT "TEMPO PARA GRAVACAO=";T1/60;"s"
126 PRINT "TEMPO PARA LEITURA =";T2/60;"s"
128 END
```

Repita o ensaio com outros tamanhos de registros e monte uma tabela comparativa do tipo:

TAMANHO DO REGISTRO (BYTES)	TEMPO GASTO POR REGISTRO	
	NA GRAVACÃO	NA LEITURA
10		
40		
70		
100		
130		

O programador verá que os tempos de acesso ao disco são relativamente "grandes" quando comparados aos tempos de manipulação de dados na memória. Porisso é importante escolher um algoritmo de ordenação que efetue o mínimo de leituras e gravações.

Na ordenação de um Cadastro de Fornecedores, por exemplo, estarão em jogo as seguintes informações:

- Nome	20
- Endereço	50
- CEP	5
- Cidade	17
- Estado	2
- Valor	4
- Telefone	7
- C.G.C.	14
- Inscrição Estadual	9

TOTAL = 128 bytes

Uma das idéias seria a de se efetuar a ordenação totalmente na memória. Para isso, o programa teria que inicialmente, transferir todo o conteúdo do arquivo para a memória, em variáveis do tipo NOME$(N), EDR$(N), etc., depois efetuar a ordenação propriamente dita e, terminada a ordenação, gravar de volta no disco, as informações já ordenadas.

```
100 ' --------------------
102 ' Ordenacao na Memoria
104 ' --------------------
106 OPEN "ARQUIVO" AS #1 LEN=128
108 FIELD #1,20 AS N$,50 AS E$,5 AS C$,
               17 AS M$,2 AS F$,4 AS V$,
               7 AS T$,14 AS G$,9 AS I$
110 FIELD #1,128 AS X$
112 N = LOF(1) / 128
114 DIM G$(N)
116 FOR REG=1 TO N
118   GET 1;REG
120   G$(REG) = X$
122 NEXT REG
126 K = N
130 X = 0
132 ZL =CVS(MID$(G$(1),95,4))
134 FOR J=2 TO K
136   ZH = CVS(MID$(G$(J),95,4))
138   IF ZL<ZH THEN SWAP G$(J),G$(J-1):
                          X = 1:
                          ELSE ZL=ZH
140 NEXT J
142 K = K - 1
144 IF X=1 AND K>2 THEN GOTO 130
 .
 .
 .
```

A vantagem desse procedimento é que será efetuada apenas uma única leitura e uma única gravação dos registros, mas, na maioria das aplicações, iremos deparar com o problema da área livre na memória para podermos armazenar todas as informações.

No Cadastro acima, cada registro acupa 128 bytes. Essas informações, quando definidas na memória, na forma de strings, passarão a ocupar 3 bytes a mais por variável, o que fará com que cada registro de 128 bytes, quando transferido para a variável G$, ocupe (128 + 3) = 131 bytes na memória.

Vamos calcular a quantidade de registros que poderão ser armazenados na memória.

Supondo que a memória do seu computador, descontado o espaço ocupado pelo programa de ordenação, disponha de 20.000 bytes livres, o que pode ser constatado pelo comando PRINT FRE(""), a quantidade máxima será de 20000/131 = 152 registros na memória, o que é muito pouco para a maioria das aplicações.

Uma das saídas seria a de se aumentar o tamanho da memória do seu computador, mas isso não é fácil nem barato.

Resta, então, a alternativa de ordenação indireta, armazenando apenas as informações que farão parte da chave de ordenação. Nesta alternativa, o programa deve efetuar os seguintes passos:

1 - leitura de todos os registros, definindo, na memória, uma variável com as informações que compõem apenas a chave de ordenação.

2 - utilizar um **algoritmo** que ordene um vetor de posição, que após a ordenação, possua a posição relativa dos registros.

3 - Trocar as posições relativas dos registros no arquivo, lendo o registro em um endereço e gravando-o no endereço indicado pelo vetor de ordenação.

Nesta alternativa, a área livre da memória será usada para o armazenamento somente da chave de ordenação e não para o registro todo. Para a ordenação de 500 registros segundo o número de telefone, necessitaremos de (7 + 3) * 500 = 5.000 bytes na memória.

Se a chave for constituída de CEP + TELEFONE, necessitaremos de (5 + 7 + 3) * 500 = 7.500 bytes na memória.

Se a chave for constituída pelo endereço, necessitaremos de (50 + 3) * 500 = 26.500 bytes, o que, evidentemente, só teremos quando a memória for incrementada com mais módulos de expansão.

Esgotadas as alternativas de ordenação total ou parcial na memória, restarão as alternativas de ordenação direta no disco.

A técnica de se efetuar a ordenação direta no disco, isto é, comparando 2 registros de cada vez e trocando ou não suas posições a cada comparação, deve ser utilizada somente em casos extremos, pois esta técnica irá promover uma grande troca de posições dos registros, e isso irá diminuir, consideravelmente, a confiabilidade nos dados, tendo em vista que em cada gravação, existe o risco de o cabeçote de gravação não conseguir gravar a informação com a fidelidade neces-

sária, além de aumentar consideravelmente o tempo de ordenação por causa da grande quantidade de leituras e gravações que serão necessárias.

Antes de partir para esse caso extremo, o programador deve pesquisar a eficiência de outras alternativas, tais como dividir o arquivo em vários arquivos menores, ordenar individualmente cada um deles e, após as ordenações, juntar novamente todos esses arquivos em um único.

Por exemplo, desejando-se ordenar um Cadastro com 1.000 Funcionários, tendo cada registro o comprimento de 67 bytes, iremos necessitar de um espaço na memória de (67 + 3) x 1.000 = 70.000 bytes, o que tornaria inviável a ordenação na memória. Porém, se esse arquivo for dividido em 4 outros arquivos menores, cada um desses outros arquivos irá ter em média

1.000 / 4 = 250 registros,

de modo que o espaço necessário na memória, para a ordenação de cada um desses arquivos separadamente, será de (67 + 3) x 250 = 17.500 bytes, o que torna viável a ordenação.

Vejamos os passos dessa técnica:

1 - Analisar os Nomes no Cadastro, tentando estimar a distribuição percentual da primeira letra dos nomes. Assim, descobrimos, por exemplo, que 25% dos funcionários tem o nome começando com as letras de A a E, mais 25% com as letras de F a L e 25% com as letras de M a P.

2 - Executar um programa que, a partir do Cadastro, crie 4 novos arquivos, para onde serão transferidos os dados cadastrais, em função da primeira letra do nome.

```
90  MAXFILES = 5
100 OPEN "A:CADASTRO.ARQ" AS #1 LEN=67
102 FIELD #1,67 AS G$
104 REG = LOF(1) / 67
106 OPEN "A:PARTE.A" AS #2 LEN=67
108 FIELD #2,67 AS GA$
110 OPEN "A:PARTE.B" AS #3 LEN=67
112 FIELD #3,67 AS GB$
114 OPEN "A:PARTE.C" AS #4 LEN=67
```

```
116 FIELD #4,67 AS GC$
118 OPEN "A:PARTE.D" AS #5 LEN=67
120 FIELD #5,67 AS GD$
122 GA = 0
124 GB = 0
126 GC = 0
128 GD = 0
130 FOR N=1 TO REG
132   GET #1,N
134   IF LEFT$(G$,1)<="E" THEN GA=GA+1 :
          PUT #2,GA : GOTO 142
136   IF LEFT$(G$,1)<="L" THEN GB=GB+1 :
          PUT #3,GB : GOTO 142

138   IF LEFT$(G$,1)<="P" THEN GC=GC+1 :
          PUT #4,REGC : GOTO 142
140   GD = GD+1 : PUT #5,GD
142 NEXT N
144 CLOSE
146 END
```

3 - Executar o programa de ordenação na memória em cada um desses 4 arquivos criados.

4 - Juntar todos os 4 arquivos em um único.

Dos inúmeros algoritmos disponíveis para a ordenação na memória, podemos empregar um, muito conhecido, e amplamente empregado, que consiste em se comparar os elementos do conjunto, 2 de cada vez, por exemplo CHAVE(m) e CHAVE(n). Nos casos em que CHAVE(m) for maior que CHAVE(n), um vetor de posição ORDEM(m) deve ser incrementado, indicando que o elemento m, sendo maior que o elemento n, deve ocupar uma ordem maior, o que será representado pelo fato de ORDEM(m) ser maior que ORDEM(n).

O núcleo do algoritmo será o seguinte:

```
100 FOR M=1 TO REG
102   FOR N=1 TO REG
104     IF CHAVE(M)<CHAVE(N) THEN
            ORDEM(N)=ORDEM(N)+1
106   NEXT N
108 NEXT M
```

Vejamos um exemplo prático do funcionamento desse algoritmo:

Sejam 5 números dados na sequência: 47, 87, 13, 86 e 10. Vamos dispor esses números segundo uma tabela, para que possamos acompanhar, etapa por etapa, o desenvolvimento do algoritmo. Em cada etapa, um número é comparado com os demais e nos casos em que o número que se compara for menor que o número comparado, o vetor ORDEM, correspondente ao número comparado é incrementado:

	47	87	13	86	10
1 - inicialmente, o vetor ORDEM contém zeros:	0	0	0	0	0
2 - comparar o 47 com todos:	0	1	0	1	0
3 - comparar o 87 com todos:	0	1	0	1	0
4 - comparar o 13 com todos:	1	2	0	2	0
5 - comparar o 86 com todos:	1	3	0	2	0
6 - comparar o 10 com todos:	2	4	1	3	0

Após essas comparações, o seguinte comando dará os números, na sequência inicial e na sequência ordenada:

```
200 FOR I=1 TO REG
202   SAIDA(ORDEM(I)+1) = CHAVE(I)
204 NEXT I
200 FOR I=1 TO REG
202   PRINT CHAVE(I),SAIDA(I)
204 NEXT I
```

O vetor ORDEM(n) contém, em cada elemento, a posição relativa dos elementos da CHAVE após a ordenação. Obtido o vetor ORDEM, o programa poderá adotar uma das duas situações seguintes:

1 - Utilizar o vetor ORDEM para poder, por exemplo, imprimir um relatório onde os itens apareçam em ordem, mas não alterar a ordem relativa dos registros no arquivo.

2 - Utilizar o vetor ORDEM para efetivamente alterar a posição relativa dos registros no arquivo, de modo que o arquivo ficará ordenado.

Na primeira situação, os registros do arquivo continuarão na mesma sequência em que se encontravam. A desvantagem é que se for necessário imprimir mais uma vez o relatório, os registros precisarão ser novamente ordenados, enquanto que a vantagem se resume em não se ter riscos de erros de gravações, visto que o arquivo é acessado somente para leituras.

Na segunda situação, as posições relativas dos registros são alteradas, isto é, primeiro registro será gravado no lugar do terceiro que por sua vez será gravado no lugar do segundo, este no quinto e finalmente, este no primeiro lugar, obrigando a uma leitura e a uma regravação de praticamente todos os registros do arquivo.

Uma rotina que reorganiza os registros de um arquivo, em função da sequência contida em um vetor ORDEM, tem o seguinte aspecto:

```
90  MAXFILES = 2
100 OPEN "ORIGEM" AS #1 LEN=56
102 FIELD #1,56 AS GO$
104 OPEN "DESTINO" AS #2 LEN=56
106 FIELD #2,56 AS GD$
108 REG = LOF(1) / 56
110 FOR N = 1 TO REG
112   GET #1,N
114   LSET GD$ = GO$
116   PUT #2,ORDEM(N)+1
118 NEXT N
```

Como último recurso, esgotadas as tentativas de ordenação na memória, restará a alternativa de ordenação direta no disco.

Um algoritmo de ordenação direta no disco é apresentado a seguir:

```
100 '------------------
102 'Ordenacao no Disco
104 '------------------
106 OPEN "ARQUIVO" AS #1 LEN=128
108 FIELD #1,94 AS A$,4 AS Z$,30 AS D$
110 FIELD #1,128 AS T$
114 K = LOF(1) / 128
120 X = 0
122 GET #1,1
124 ZL = CVS(Z$)
```

```
126 TL$ = T$
128 FOR J=2 TO K
130   GET #1,J
132   ZH = CVS(Z$)
134   TH$ = T$
136   IF ZL<=ZH THEN ZL  = ZH  :
                     TL$ = TH$ :
                     GOTO 148
138   LSET T$ = TL$
140   PUT #1,J
142   LSET T$ = TH$
144   PUT #1,J-1
146   X = 1
148 NEXT J
150 K = K - 1
152 IF X=1 AND K>2 THEN GOTO 120
154 CLOSE #1
156 END
```

ORDENAÇÃO DE ARQUIVOS SEQUENCIAIS

Tudo aquilo que foi visto nos parágrafos anteriores, só se aplica a arquivos randômicos, pois os comandos de acesso direto a registros específicos do tipo:

```
105 GET #1,N
```

ou:

```
107 PUT #2,ORDEM(I)+1
```

só são possíveis em arquivos randômicos.

Então, os arquivos sequenciais não podem ser ordenados?

A priori, a resposta seria NÃO, pois não existe comando do BASIC para o acesso direto aos registros de arquivos sequenciais, de modo que não é possível acessar, aleatoriamente, um determinado registro do arquivo. Mesmo que existisse tal comando, os registros possuem tamanhos diferentes e como iremos "trocar" de posição registros de tamanhos diferentes? O registro menor cabe no lugar do registro grande, mas o contrário será impossível.

A solução mais prática é a de se transformar o arquivo sequencial em um arquivo randômico, classi-

ficar o arquivo randômico e transformá-lo, por fim, em arquivo sequencial novamente.

```
100 MAXFILES = 2
101 OPEN "ARQUIVO.SEQ" FOR INPUT AS #1
102 OPEN "ARQUIVO.RND" AS #2 LEN=70
104 FIELD #2,6 AS D$,4 AS C$,2 AS Q$,8 AS P$,50 AS H$
106 N = 0
108 INPUT #1,DAT$,COD$,QUAN,PRECO,HIS$
110   LSET D$ = DAT$
112   LSET C$ = COD$
114   LSET Q$ = MKI$(QUAN)
116   LSET P$ = MKD$(PRECO)
118   LSET H$ = HIST$
119 N = N + 1
120 PUT #2,N
122 IF NOT EOF(1) THEN GOTO 108
124 REG = N
126 CLOSE #1
128 GOSUB xxxx :'Rotina de ordenação.
129 OPEN "ARQUIVO.SEQ" FOR OUTPUT AS #1
130 FOR N = 1 TO REG
132   GET #2,N
134   PRINT #1,D$,C$,CVI(Q$),CVD(P$),H$
136 NEXT N
138 CLOSE #1
140 CLOSE #2
142 END
```

TÉCNICAS DE ACESSO AOS ARQUIVOS

O acesso aos arquivos é uma atividade que consiste na leitura dos registros a procura de uma determinada informação que foi armazenada anteriormente em algum registro do arquivo.

Por exemplo, sabemos que no Cadastro de Clientes, existe um registro com o endereço do Sr. Joaquim de Oliveira. Dada a informação "Joaquim de Oliveira", qual é o procedimento que o programa deve seguir para encontrar o endereço do mesmo?

O acesso aos arquivos é um assunto que muitas vezes não tem muito a ver com a forma de organização do arquivo, isto é, se sequencial ou randômico.

A rotina poderá procurar o registro desejado, lendo um a um todos os registros do arquivo até encontrar o registro procurado ou aplicar alguma técnica que permita "adivinhar" o número do registro ou pelo menos a região onde ele se encontra.

Vamos analisar os problemas de acesso, fazendo uma analogia com um arquivo de fichas de papelão.

Vamos supor que tenhamos em mãos um arquivo de clientes montado com fichas de papelão, e vamos supor também que essas fichas estão organizadas por ordem alfabética do Nome.

Situação 1: Qual é o cliente que reside na rua Vergueiro, 102 ?

Se você não conhece os dados do arquivo, isto é, não tem nenhuma "dica" de por onde começar, a única maneira de encontrar tal ciente, será consultar ficha por ficha, confrontando a Informação "rua Vergueiro, 102" com o endereço que consta em cada ficha, até encontrar a informação desejada ou até encontrar o final do fichário.

Nesta situação, serão acessados em média N/2 fichas.

Situação 2: Qual é o cliente mais antigo?

Como as fichas estão em ordem alfabética, a ficha do cliente mais antigo pode estar em qualquer posição.

Neste caso, a procura é um pouco mais complexa que a procura apresentada na situação anterior, pois não temos nem mesmo a informação que deverá ser confrontada com a data de registro que consta nas fichas.

A não ser que seja utilizada alguma técnica "especial" em que seja possível determinar a priori a ficha procurada, por exemplo pela cor um tanto quanto amarelada da ficha por causa da idade da mesma, a procura deve ser realizada em 2 etapas, objetivando responder as seguintes perguntas:

1 - Qual a data mais antiga?
2 - Qual o cliente com essa data?

Para que a procura seja viável, devemos fazer determinadas suposições, por exemplo, supor, inicial-

mente, que o cliente mais antigo seja aquele da primeira ficha, elegendo, desse modo, a data de cadastramento desse cliente como sendo a mais antiga.

Feito isso, comparar essa data com a data de cadastramento do segundo cliente. Se essa nova data for anterior aquela adotada, passar a considerar como data mais antiga a data de cadastramento desse segundo cliente.

Repetindo-se esse procedimento até o último registro do arquivo, saberemos qual é a data mais antiga, satisfazendo a primeira parte do problema.

Nesta situação serão acessadas, em média, N fichas.

Situação 3: Qual é a idade do Silvio de Oliveira ?

Sabendo que o fichário está em ordem alfabética, a ficha do Silvio não está, com certeza, na primeira metade do fichário. Neste caso, a procura deve ser iniciada a partir da metade do fichário.

Caso não se encontre o nome "Silvio de Oliveira", a procura pode ser encerrada antes do final do fichário, pois sabemos que o mesmo está em ordem alfabética.

Nesta situação, serão acessadas, em média, uma certa quantidade de fichas dependendo do "bom senso" daquele que manuseia o fichário.

Como se vê, existem no dia a dia, muitas situações diferentes de acesso às informações de um arquivo e para cada uma delas existe uma técnica apropriada.

Qual a importância em se analisar as técnicas de acesso?

Ler um registro de um arquivo, significa transferir uma certa quantidade de bytes do disco para a memória.

Como se sabe, a conexão do drive do disco com a memória é feita por intermédio de um complexo circuito eletrônico que transmite byte a byte a uma certa velocidade.

Isso implica em dizer que o tempo gasto para a leitura de um registro depende do tamanho do registro.

Infelizmente, não existe uma maneira de ler apenas uma parte do registro. Em todas as leituras é preciso ler o registro completo.

Desse modo, desejando-se diminuir o tempo gasto na busca de uma informação, a única saída é tentar, de alguma forma, diminuir a quantidade de leituras no arquivo.

A análise da estrutura de arquivo juntamente com as maneiras possíveis de acessar as informações nele contidas deve resultar em um algoritmo de busca em que a quantidade média de leituras em disco seja diminuida ao máximo.

Vejamos algumas das técnicas mais empregadas:

ACESSO SEQUENCIAL A ARQUIVOS

A leitura dos registros é efetuada em sequência, a partir do primeiro regitro, até encontrar o registro desejado ou o final do arquivo.

```
100 OPEN "MOVTO.ARQ" FOR INPUT AS #1
102 INPUT "Qual o nome ";N$
200 INPUT #1,NOME$,ENDE$,CIDADE$,ESTADO$
202 IF EOF(1) THEN GOTO 400
204 IF NOME$=N$ THEN GOTO 300
206 GOTO 200
300   PRINT "*** ENCONTREI ***"
302   PRINT "nome = ";NOME$
304   PRINT "endereco = ";ENDE$
306   PRINT "cidade = ";CIDADE$
308   PRINT "estado = ";ESTADO$
310 END
400 PRINT "*** NAO ENCONTREI ***"
402 END
```

4.6.2 - Acesso Sequencial em arquivo randômico.

A leitura dos registros é efetuada em sequência, a partir do primeiro regitro, até encontrar o registro desejado ou o final do arquivo.

```
100 OPEN "MOVTO.ARQ" AS #1 LEN=67
102 FIELD #1,20 AS N$,30 AS E$,15 AS C$,2 AS F$
104 REG = LOF(1) / 67
106 INPUT "Qual o nome ";NOME$
200 FOR N=1 TOREG
202   GET #1,N
204   IF NOME$=N$ THEN GOTO 400
206 NEXT N
```

```
300 PRINT "*** NAO ENCONTREI ***"
302 END
400   PRINT "*** ENCONTREI ***"
402   PRINT "nome = ";N$
404   PRINT "endereco = ";E$
406   PRINT "cidade = ";C$
408   PRINT "estado = ";F$
410 END
```

Lembrando sempre que a rotina de procura geralmente não faz parte do núcleo principal do programa, desenvolve-la a parte para ser acessada por um comando GOSUB.

```
1400 ' ----------------------------------------
1402 ' ROTPMOV.FNT = Rotina Procura um Movimento
1404 ' ----------------------------------------
1406 ' Esta rotina permite procurar um
1408 ' determinado registro no arquivo
1410 ' de Movimento.
1412 ' R.M.Watanabe               20/04/87
1416 REG% = 0
1420 REG% = REG% + 1
1422 IF REG% > MV% THEN REG%=0:RETURN
1424 GET #NARQ,REG%
1426 IF A$ = MD$ AND TIPO$ = MV$ THEN RETURN
1428 GOTO 1420
```

```
SAVE "ROTPMOV.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

ACESSO DIRETO EM ARQUIVO RANDÔMICO

A leitura dos registros é efetuada diretamente no registro desejado. Para isso é necessário conhecer o número do registro.

```
100 OPEN "MOVTO.ARQ" AS #1 LEN=67
102 FIELD #1,20 AS N$,30 AS E$,15 AS C$,2 AS F$
104 MAX = LOF(1) / 67
106 INPUT "Qual o numero do registro ";REG
108 IF REG>MAX THEN PRINT"FORA DO ARQUIVO" :
```

```
                  GOTO 106
202 GET #1,REG
204 PRINT "nome = ";N$
206 PRINT "endereco = ";E$
208 PRINT "cidade = ";C$
210 PRINT "estado = ";F$
212 END
```

BUSCA BINÁRIA EM ARQUIVO RANDÔMICO

A leitura dos registros é efetuada dividindo-se o arquivo em duas partes, daí o nome da técnica, e acessando-se a parte onde haja maior probabilidade de encontrar o registro desejado.

Para que isso seja possível, o arquivo deve estar ordenado pelo campo onde é feita a procura.

A grande vantagem da busca binária reside no fato de a quantidade de leituras necessárias ser bem pequena, pois enquanto que no acesso sequencial lê-se em média N/2 registros, na busca binária esse número cai para log2N, isto é, para um arquivo com 5.000 registros, o acesso sequencial irá ler 2500 registros em média, enquanto que a busca binária irá ler no máximo 17 registros.

```
100 OPEN "CADASTRO.ARQ" AS #1 LEN=67
102 FIELD #1,20 AS CD$,15 AS C$,2 AS F$
104 MX% = LOF(1) / 67
106 INPUT "Qual o nome ";A$
200 GOSUB 1200
202 IF C%=0 THEN PRINT "*** NAO ENCONTREI ***" :
                 GOTO 106
204 PRINT "*** ENCONTREI ***"
206 PRINT CD$
208 PRINT E$
210 PRINT C$
212 PRINT F$
214 END


          MERGE "ROTPROC.FNT"
```

```
1200 ' ------------------------------------------
1201 ' ROTPROC.FNT = Procura Material no Cadastro
1202 ' ------------------------------------------
1203 ' Se encontrar, C% sai com o n. do registro.
1204 ' Se nao encontrar, FIRST% volta com o n. do
1205 ' registro onde devera ser encaixado  o novo
1206 ' registro.
1207 '
1210 FIRST% = 1
1212 LAST%  = MX%
1220    C% = INT((FIRST%+LAST%)/2)
1221    IF C%=0 THEN GOTO 1232
1222    GET #NARQ,C%
1224    IF A$=CD$ THEN GOTO 1234
1226    IF A$>CD$ THEN FIRST%=C%+1
1228    IF A$<CD$ THEN LAST%=C%-1
1230    IF FIRST$<=LAST% THEN GOTO 1220
1232 C% = 0
1234 RETURN
```

```
SAVE "ROTPROC.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

Não custa lembrar que as rotinas de procura devem, na medida do possível, serem programadas fora do núcleo principal do programa.

BUSCA INDEXADA EM ARQUIVO RANDÔMICO

Nesta técnica, trabalhamos com 2 arquivos distintos.

O primeiro deles, é o arquivo principal e contém todos os dados. O outro, é o arquivo de índices e contém somente o campo de procura e o número de registro correspondente.

Vejamos como isso funciona, com um exemplo:

Vamos supor que o arquivo contenha o Cadastro de Funcionários de uma empresa. Ora, tal tipo de arquivo é um arquivo que consideramos "largo", pois cada registro ocupa uma quantidade considerável de bytes, por causa do grande volume de dados.

CAMPOS	QUANTIDADE DE BYTES
- Nome do Funcionário	30
- Endereço	50
- CEP	5
- Cidade	20
- Estado	2
- Data do Nascimento	6
- Número do R.G.	7
- Número do CPF	11
- Número da CTPS	9
- Número do PIS	11
- Código do Banco para Pagamento	3
- Código do Banco para FGTS	3
- Quantidade de Dependentes	2
- Data da Admissão	6
- Número de Registro (Chapa)	5
- Código da Seção	4
- Código do Cargo	4
- Salário Inicial	4
- Salário Atual	4
TOTAL =	181 bytes

Na emissão da Folha de Pagamento, por exemplo, para cada funcionário, o programa deverá consultar o cadastro para saber o nome do funcionário, entre outras coisas, pois todas as digitações de dados e cálculos são realizados apenas pelo número de registro.

Ora, se a empresa tem 1.000 funcionários, isso significa que o programa irá acessar 1.000 vezes esse cadastro.

Se o programa utilizar a técnica de acesso sequencial, pelo fato de no acesso sequencial ser feita N/2 = 500 leituras a cada acesso, o programa irá realizar um total de 1.000 X 500 = 500.000 leituras.

Se o programa utilizar a técnica de busca binária, pelo fato de na busca binária ser feita log2N=7 leituras a cada acesso, o programa irá realizar um total de 1.000 X 7 = 7.000 leituras.

Comparando-se os dois tipos de acesso acima, vê-se que a segunda técnica representa uma economia sensível.

Mas podem ocorrer situações em que mesmo essas 7.000 leituras, devido a largura do registro (181

bytes) chegue a consumir um tempo relativamente longo, pois o tempo de leitura de um registro é proporcional à quantidade de bytes que serão transferidos do disco para a memória.

No caso em questão, deverão ser transferidos 7.000 X 181 = 1.267.000 bytes.

Para se poder utilizar a busca indexada, devemos criar um segundo arquivo contendo:

CAMPOS	QUANTIDADE DE BYTES
- o número de Chapa	5
- o número do registro no cadastro	4

Este arquivo cujo registro ocupa somente 9 bytes será conhecido como arquivo de índices.

Para se localizar o nome do funcionário cujo número de chapa seja, por exemplo, 437, o programa deverá:

1 - Acessar os registros do arquivo de índices, aplicando a busca binária, até encontrar o funcionário de número 437. Nesse acesso serão transferidos em média 7 X 9 = 63 bytes.

2 - Encontrado o funcionário procurado, pegar o número do registro do mesmo no Cadastro.

3 - Acessar diretamento o registro no Cadastro. Nesse acesso serão transferidos os 181 bytes do funcionário.

Como se vê, na busca indexada serão transferidos, para cada funcionário, 63+181=244 bytes ou, se considerarmos a emissão total dos 1.000 funcionários, 244.000 bytes, o que é bem inferior aos 1.267.000 bytes da alternativa anterior.

O uso de 2 arquivos que embora distintos se referem ao mesmo conjunto de dados, requerem certos cuidados:

1 - Toda vez que houver uma inclusão ou exclusão de algum registro no Cadastro, o arquivo de índices deve ser alterado também.

2 - O arquivo de índices deve ser sempre ordenado.

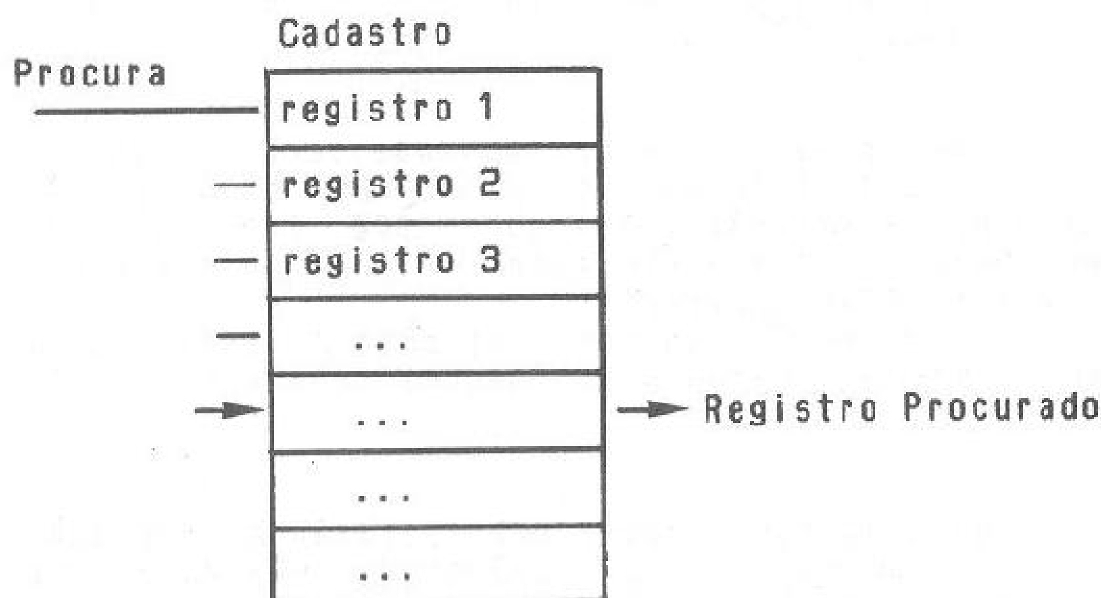
Mas, em compensação, apresenta algumas vantagens:

1 - Os registros do arquivo principal podem estar ordenados de qualquer forma, pois quem garante o ordenamento é o arquivo de índices. Essa é a grande vantagem da busca indexada, pois, havendo a necessidade de se incluir um novo funcionário no Cadastro o registro correspondente a esse novo funcionário poderá ser simplesmente adicionado no fim do arquivo principal.

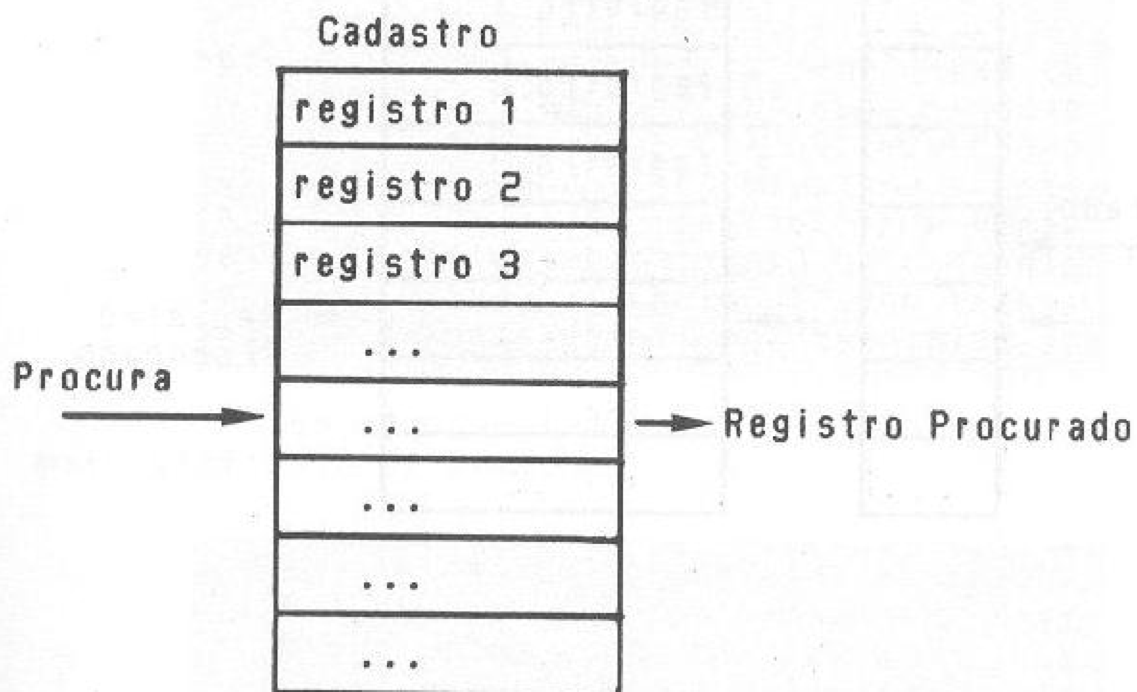
2 - Pode-se ter vários arquivos de índices, um ordenado pelo número de Chapa, outro pelo nome do funcionário, etc. Na situação de se desejar o salário, tendo-se o número de chapa, bastará acessar o arquivo de índices ordenado pelo número de chapa, enquanto que na situação de se desej r o salário tendo-se o nome do funcionário, bastará acessar o outro arquivo, isto é, o arq ivo de índices ordenado pelo nome de funci nários.

A seguir são representados os esquemas de acesso das várias técnicas de acesso apresentadas.

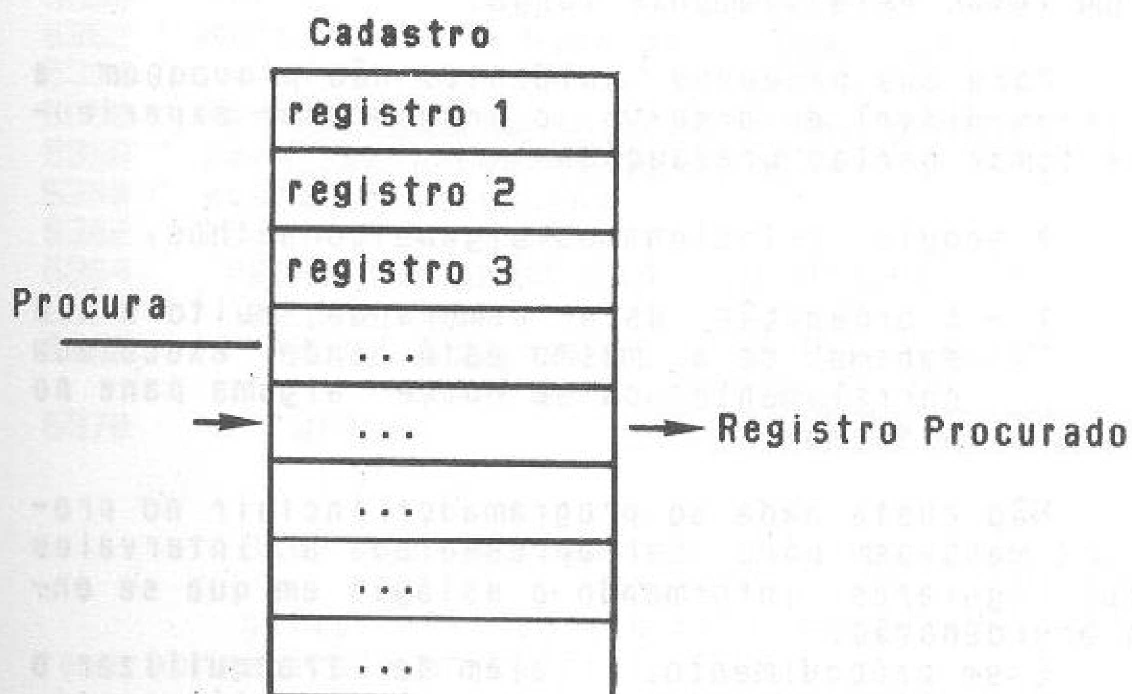
ACESSO SEQUENCIAL



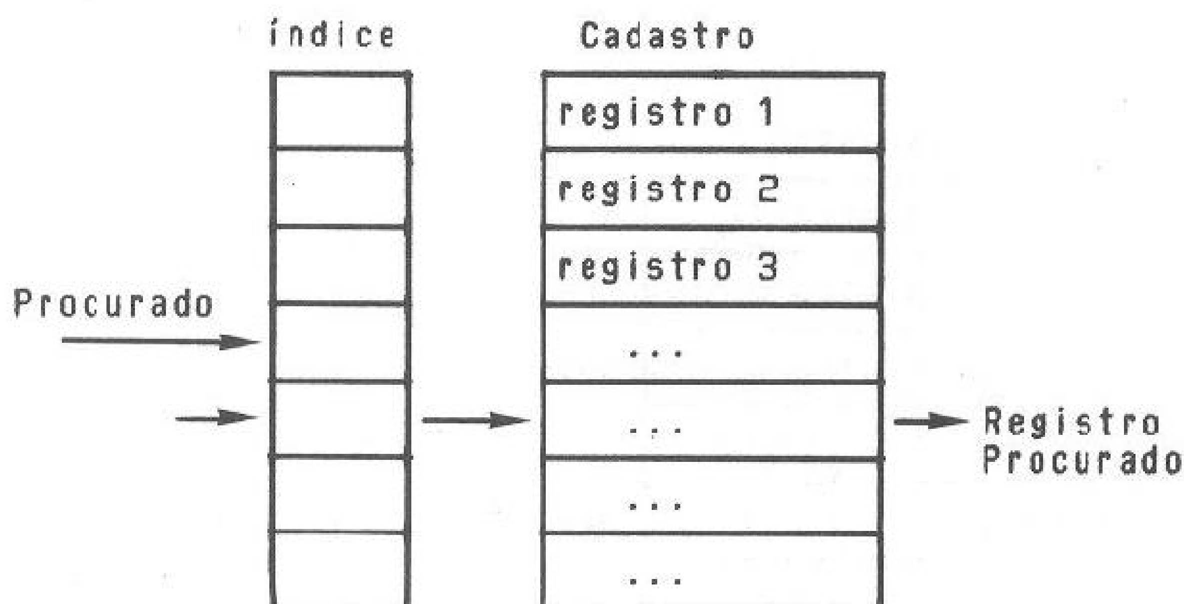
ACESSO DIRETO



BUSCA BINÁRIA



BUSCA INDEXADA



CUIDADOS ESPECIAIS

Nas operações de ordenação de arquivos, estaremos operando com vários arquivos em disco, e tais operações, dependendo do tamanho de cada registro e da quantidade total de registros no arquivo, poderão consumir um tempo relativamente longo.

Para que pequenos acidentes não provoquem a perda irremediável do arquivo, o programador experiente deve tomar certas precauções.

A seguir, relacionamos alguns conselhos:

1 - A ordenação está demorando muito e não sabemos se a mesma está sendo executada corretamente ou se houve alguma pane no sistema.

Não custa nada ao programador incluir no programa uma mensagem para ser apresentada a intervalos de tempo regulares, informando o estágio em que se encontra a ordenação.

Esse procedimento, além de tranquilizar o operador, dando-lhe a certeza de que tudo está corren-

do bem, permite uma avaliação do tempo que a ordenação ainda vai demorar. Por exemplo, podemos produzir uma tela como a mostrada a seguir.

```
Já foram ordenados 10 registros.
Já foram ordenados 20 registros.
Já foram ordenados 30 registros.
Já foram ordenados 40 registros.
Já foram ordenados 50 registros.
Já foram ordenados 60 registros.
Já foram ordenados 70 registros.
Já foram ordenados 80 registros.
              •
              •
              •
```

Caso, o programador deseje apresentar algo mais profissional como:

```
ORDENAÇÃO: <<<<<<<<<..............
           0%        50%        100%
```

Poderá embutir no programa a seguite rotina:

```
5350 ' -----------------------------------
5352 ' MAPAACC.FNT = Mapa de Acompanhamento
5354 ' -----------------------------------
5356 ' Mostra nas linhas 17 e 18 uma escala
5358 ' para ser apresentado o estagio de
5360 ' evolucao de ciclos.
5362   LOCATE 5,17
5364    PRINT "0%"SPACE$(13)"50%"SPACE$(11)"100%"
5366     LOCATE 5,18
5368      PRINT CHR$(124);:FOR N=1 TO 6:PRINT SPACE$(4)
                CHR$(124);:NEXT N
5370       RETURN
```

SAVE "MAPAACC.FNT",A

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

Naturalmente, para que o mapa possa ser utilizado por diversos trechos do programa, a parte que desenha o mapa inicial deve ser programada fora do núcleo principal do programa.

2 - Durante a ordenação de um arquivo, poderá ocorrer alguma pane no sistema provocando a perda do arquivo.

Para prevenir-se contra essas situações desagradáveis, criar um arquivo provisório, cópia do arquivo principal, e executar a ordenação em cima do arquivo provisório.

Para automatizar esses procedimentos, incluir os comandos dentro do programa.

```
500 '---------
502 'ORDENACAO
504 '---------
506 COPY "CADASTRO.ARQ" TO "ARQUIVO.PRV"
508 OPEN "ARQUIVO.PRV" AS #1 LEN=57
 .
 .
 .
```

Após a ordenação, caso não tenha ocorrido nenhum erro, eliminar o arquivo principal e trocar o nome do arquivo provisório:

```
614 KILL "CADASTRO.ARQ"
616 NAME "ARQUIVO.PRV" AS "CADASTRO.ARQ"
```

3 - O arquivo provisório deve ter um nome sugestivo, como:

ARQUIVO.PRV

pois se houver esquecimento em eliminá-lo do disco, e posteriormente o usuário estiver analizando o conteúdo com o comando FILES, saberá, imediatamente que se trata de um arquivo provisório e assim poderá eliminá-lo sem medo, com o comando KILL.

4 - Ao criar, dentro do programa, um novo arquivo, verifique se já não existe um outro arquivo com o mesmo nome. Lembre-se que em situações em que o sistema operacional encontra no disco um arquivo com o mesmo nome, os dados serão gravados por cima, perdendo-se aquilo que estava anteriormente no arquivo.

A função LOF permite saber a quantidade de bytes ocupados pelo arquivo. Um arquivo cujo comprimento seja diferente de ZERO é um arquivo que já existe.

```
314 PRINT "Vou abrir um arquivo novo"
315 OPEN "ARQUIVO.ARQ" FOR APPEND AS #1
316 IF LOF(1)>0 THEN PRINT "Ja existe um outro
    arquivo com o mesmo nome"
```

O comando ONERROR=65 permite saber se o arquivo já existe:

```
211 ON ERROR GOTO 600
212 OPEN "ARQUIVO.ARQ" FOR APPEND AS #1
      .
      .
      .
600 IF ERR=65 THEN PRINT "Ja existe um outro arquivo
                          com o mesmo nome."
```

O comando ONERROR = 53 permite saber se o arquivo não existe:

```
211 ON ERROR GOTO 600
212 OPEN "ARQUIVO.ARQ" FOR APPEND AS #1
      .
      .
      .
600 IF ERR=53 THEN RESUME
```

5 - Verifique a quantidade de bytes disponíveis no disco.

É bastante irritante, um programa de ordenação levar 2 horas para ordenar o arquivo e quando estiver quase no fim, esgotar a capacidade de armazenamento do disco.

Porisso, é bastante conveniente verificar, antes de mais nada, se há espaço suficiente para caber o novo arquivo.

```
611 IF DSKF(1)*1024<REG*67 THEN PRINT "Nao ha espaco
                suficiente neste disco, coloque outro
                menos cheio."
```

6 - Não esqueça de fechar a porta do drive após inserir o disco.

A atitude de fechar a porta do drive, geralmente é automática e nem sempre pensamos nisso, existindo o risco de nos esquecermos.

Imagine que o programa gaste um tempo relativamente grande para ordenar os registros de um arquivo que esteja no drive A: e após esse tempo ele irá gravar no drive B:, um novo arquivo, com os registros já ordenados, e, se nessa hora a porta do drive estiver aberta, o sistema operacional irá acusar tal esquecimento com uma mensagem do tipo:

Erro de Disco

e todo o trabalho de ordenação estará perdido.

Para evitar esse tipo de problema, faça com que o próprio programa verifique a existência ou não de um disco no drive B:, com algo do tipo:

```
211 ON ERROR GOTO 600
212 OPEN "ARQUIVO.ARQ" FOR OUTPUT AS #1
         .
         .
         .
600 IF ERR=68 THEN PRINT "Disco protegido."
602 IF ERR=70 THEN PRINT "Falta disco no drive."
```

Esses códigos de erro, e mais alguns relacionados em seu manual (*), são produzidos na execução do comando OPEN, quando o sistema operacional consulta o

diretório do disco. Ocorrendo uma condição anormal, o sistema operacional ativa a instrução ON ERROR e transfere o comando para a linha indicada no comando.

O leitor poderá desenvolver uma "super-rotina" que preve todas as situações que poderão ocorrer nos acessos ao disco para orientar o usuário de como proceder em cada uma delas.

7 - Nas situações em que o programa solicita ao usuário um nome para o arquivo que será criado, facilite o trabalho do usuário mostrando os arquivos já existentes no disco, com o auxílio do comado FILES: .

```
600 CLS
602 FILES "*.ARQ"
604 PRINT "Fornecer um nome para o arquivo, diferente
          dos acima (maximo 8 caracteres)."
606 INPUT N$
608 IF LEN(N$)>8 THEN GOTO 604
610 ARQ$ = N$ + ".ARQ"
612 OPEN ARQ$ AS #1 LEN=68
```

(*) Para uma relação completa dos códigos de erro consulte o manual de sua interface. No caso das iterfaces para MSX, existem 3 tipos:

MICROSOL (e seus clones tipo DDX, TROPIC, LASER..)
Manual: SISTEMA DE DISCO PARA MSX (ALEPH) - Renato da Silva Oliveira- Rubens Pereira Silva Jr

SHARP
Manual: USANDO O DISK DRIVE NO MSX (ALEPH)- Rubens Pereira Silva Jr

TECHNOAHEAD E NOR (padrão MSX internacional)
Manual: DRIVES LEOPARD DE 3 1/2" (ALEPH)- Carvalho Jr- Oliveira- Piazzi
Obs: Esta interface serve também para drives de 5 1/4"

5

técnicas de saída

Entende-se como "saída" toda e qualquer manifestação do programa, quando determinadas informações são encaminhadas a um periférico conectado ao computador.

Vejamos algumas sugestões para melhorar a perfomance dos programas.

LIMPEZA DE TELA

O comando CLS (no MSX e PC) ou o HOME (no MBASIC) apaga instantaneamente toda a tela do vídeo.

Existem situações em que desejamos apagar somente parte da tela. Para essas situações o programador poderá desenvolver uma série de rotinas, cada uma delas adequada a determinadas situações:

```
100 CLS
101 FOR N=23 TO 1 STEP -1
102   LOCATE ,N : PRINT STRING$(39,"#");
103 NEXT N
104 C = 20
105 L = 11
106 P = 0
107 FOR I = C-P TO C+P
108   LOCATE C-P,I-9 : PRINT SPC(1)
```

```
109   LOCATE C+P,I-9 : PRINT SPC(1)
110 NEXT I
111 LOCATE C-P,L-P : PRINT SPC(2*P)
112 LOCATE C-P,L+P : PRINT SPC(2*P)
113 P = P + 1
114 IF P<10 THEN GOTO 107
```

ESPAÇOS EM BRANCO E PROCESSAMENTO

Em todas as saídas formatadas, os espaços em branco entre as informações consomem tempo para serem impressas, tanto no vídeo como na impressora.

Se o programador comparar o tempo gasto para a impressão das 2 telas seguintes:

```
Antonio de Souza                 345,00
Carlos Mendes                  1.246,00
Enio Freitas                      47,30
Jose Carlos de Oliveira          412,40
Ivanildo Gomes de Carvalho        18,00
```

e:

```
  345,00 Antonio de Souza
1.246,00 Carlos Mendes
   47,30 Enio Freitas
  412,40 Jose Carlos de Oliveira
   18,00 Ivanildo Gomes de Carvalho
```

verificará que uma tabela do segundo tipo será impressa num tempo sensivelmente menor.

CARACTERES ESPECIAIS

Os caracteres especiais reduzem a velocidade de impressão.

As impressoras matriciais apresentam a característica de poderem imprimir com maiores densidades que o padrão de 10 caracteres por polegada.

Cuidado ao utilizar esse recurso, pensando em aumentar a quantidade de caracteres por linha, pois, geralmente, os caracteres do tipo comprimido reduzem em muito a velocidade de impressão.

O melhor mesmo é analisar a real necessidade das informações no relatório e imprimir somente o necessário.

TIPOS DE ESPAÇOS EM BRANCO

Existem diversas alternativas para se obter espaços em branco e, aparentemente, qualquer uma das funções abaixo pode ocasionar o mesmo efeito:

```
PRINT "      "
PRINT SPACE$(6)
PRINT SPC(6)
PRINT STRING$(6,32)
PRINT TAB(6)
```

Em relatórios extensos, pequenas diferenças de tempo ocasionadas pela escolha inadequada da função, poderão provocar um acréscimo de tempo considerável para imprimir o relatório todo.

Recomenda-se efetuar diversos testes para se conhecer os tempos. Nos testes que realizei, encontrei os tempos relativos seguintes:

FUNCÃO	TEMPO
" "	277
SPACE$(10)	280
SPC(10)	102
STRING$(32,10)	285
TAB(10)	19

O COMANDO PRINT

Os comandos PRINT e LPRINT são usados para a impressão no vídeo e na impressora respectivamente.

Existem situações em que gostaríamos de escolher o local da impressão, sem alterar o programa.

Existem também situações em que queremos enviar um relatório para a impressora mas ela está quebrada ou inoperante por algum outro motivo. A solução é "imprimí-lo" no disco.

Isso é possivel ?

SIM, isso é possível, com o uso do comando:

PRINT #

```
100 PRINT "Escolha o local da impressao:"
101 PRINT "1 - No Video"
102 PRINT "2 - Na impressora"
103 '
104 K$ = INKEY$ : IF K$="" THEN GOTO 104
105 '
106 IF K$<"1" OR K$>"2" THEN GOTO 100
107 '
108 IF K$="1" THEN SAIDA$ = "CRT:" :
               ELSE SAIDA$ = "LPT:"
110 OPEN SAIDA$ FOR OUTPUT AS #1
112 PRINT #1, "Belo Resultado"
114 END
```

Além dessa vantagem, existem situações em que o relatório deve, obrigatoriamente, ser emitido para que outros programas possam ser executados, mas, por uma incrível coincidência ou infelicidade, a impressora está quebrada.

Se o programa realiza a impressão na impressora com comandos LPRINT, o programador estará em uma bela enrascada.

Veja como conseguir uma flexibilidade ainda maior, analisando a sugestão a seguir:

```
100 PRINT "Escolha o local da impressao:"
101 PRINT "1 - No Video"
102 PRINT "2 - Na impressora"
103 PRINT "3 - No Disco"
104 K$ = INKEY$ : IF K$="" THEN GOTO 104
105 IF K$<"1" OR K$>"3" THEN GOTO 100
106 IF K$ = "1" THEN SAIDA$ = "CRT:"
107 IF K$ = "2" THEN SAIDA$ = "LPT:"
108 IF K$ = "3" THEN SAIDA$ = "ARQUIVO"
109 '
110 OPEN SAIDA$ FOR OUTPUT AS #1
111 PRINT #1,"Belo Resultado"
```

MENSAGENS DE ERRO EM PROGRAMAS

Em um programa que apresenta diversas mensagens de erro, é conveniente programar a apresentação da mensagem fora do núcleo principal do programa:

```
5300 ' ------------------------------
5302 ' ERRO.FNT = Mensagem de Erro
5304 ' ------------------------------
5306   LOCATE 0,19
5308    IF LEN(ER$)<>0 THEN PRINT ER$:
          ELSE PRINT "*** DADO INVALIDO ***"
5310     GOSUB 5010
5312      GOSUB 5200
5314       LOCATE 0,19:PRINT SPC(30)
5316        RETURN

        SAVE  "ERRO.FNT",A
```

Salvar esta rotina para ser utilizada no sistema apresentado no capítulo 9.

A FUNÇÃO USING

A função USING, utilizada nos comandos PRINT, PRINT# e LPRINT permite que os dados que serão apresentados sejam formatados conforme determinados padrões, melhorando a estética daquilo que é apresentado na tela ou na impressora.

Para que isso ocorra, basta que coloquemos entre o comando PRINT e o dado a ser impresso, a palavra USING seguida de uma string de caracteres especiais, conhecidas como máscara de edição.

Tais caracteres são os seguintes:

!	Apresenta apenas o primeiro caractere da string.
\\	Apresenta tantos caracteres da string quanto os espaços em branco entre as â.
&	Apresenta a string completa.
#	Indica as posições em que os algarismos deverão ser apresentados. Pode ser inserido 1 ponto no meio da sequência de #s, para indicar a posição do ponto decimal. Pode ser colocado um sinal + ou - no início da sequência para indicar se desejamos que o símbolo + ou - seja im-

presso na frente do número nos casos em que ele for positivo ou negativo, e o símbolo * para que os espaços na frente do número sejam preenchidos.

Exemplos:

Imprimindo o número:	com a máscara:	obtem-se:
+123.4	####.##	123.40
-123.4	####.##	-123.40
+123.4	+####.##	+123.40
-123.4	+####.##	-123.40
+123.4	-####.##	- 123.40
-123.4	-####.##	--123.40
+123.4	**####.##	***123.40
-123.4	**####.##	**-123.40
+123.4	+**####.##	***+123.40
-123.4	+**####.##	***-123.40

Existem outros símbolos que provocam formatações diferentes, como já deve ser de conhecimento do programador.

Analisando as diversas máscaras utilizadas em um programa, iremos verificar que não existe uma gama muito grande de máscaras diferentes, no mesmo programa, pois muitas delas são repetidas.

Desse modo, em vez de se usar:

```
600 PRINT "Capital de .....";  :
    PRINT USING "######.##";CAP
601 PRINT "na taxa de .....";  :
    PRINT USING "######.##";TAX
602 PRINT "amortizado em ..";  :
    PRINT USING "#########";PER
603 PRINT "resultara nas seguintes prestacoes:"
604 FOR N=1 TO PER
605   PRINT "prestacao";N;".";  :
      PRINT USING "######.##";PRE(N)
606 NEXT N
```

Pode ser usado uma forma mais elegante e econômica, do tipo:

```
600 F1$ = "######.##"
601 F2$ = "#########"
602 PRINT "Capital de .....";  : PRINT USING F1$;CAP
```

```
603 PRINT "na taxa de .....";  : PRINT USING F1$;TAX
604 PRINT "amortizado em ..";  : PRINT USING F2$;PER
605 PRINT "resultara nas seguintes prestacoes:"
606 FOR N=1 TO PER
607   PRINT "prestacao";N;".";:PRINT USING F1$;PRE(N)
608 NEXT N
```

IMPRESSÃO OTIMIZADA NA TELA

Quando estamos trabalhando com uma tela de digitação de dados, onde existem dados fixos e dados variáveis, a impressão de todos os dados na tela pode consumir alguns significativos segundos. Um digitador profissional poderá sentir essa pequena diferença e isso será o suficiente para que o programa seja classificado de "lento".

Um boa técnica seria separar a impressão dos dados fixos, da impressão dos dados variáveis.

Observe a seguinte tela de digitação:

```
  0123456789012345678901234567890123456789
 1|     Cadastramento de Funcionarios      |   tela: 01
 2|========================================|
 3|                                        |
 4|Nome:..............................     |   NF$
 5|Endereco:....................           |   EF$
 6|         ....................           |
 7|CEP:..... Cidade:....................   |   EC$,EM$
 8|Estado:.. Nascimento:../../..           |   EE$,NT$
 9|RG:......... CIC:.........-..           |   GF$,FF$
10|CTPS:...... serie:.. emissao:../../..   |CT$,ST$,DT$
11|                                        |
12|Registro:..... em:../../..              |   NR$,DR$
13|cargo:....................              |   CF$
14|secao:....................              |   SF$
15|salario:..........,..                   |   SL$
16|                                        |
17|PIS:....................                |   NP$
18|banco:... agencia:.... data:../../..    |BP$,AP$,DP$
19|dependendentes:..                       |   ND$
20|                                        |
21|FGTS:..                                 |   FG$
22|banco:... agencia:.... data:../../..    |BG$,AG$,DG$
23|                                        |
24|                                        |
```

Observe a listagem a seguir onde os dados variáveis são impressos em uma parte distinta da impressão dos dados fixos:

```
200 ' ---------------------------
202 ' Inicializacao de Variaveis
204 ' ---------------------------
222 CLEAR 1000
230 NF$ = STRING$(30,".")
240 EF$ = STRING$(20,".")
250 EC$ = STRING$(05,".")
251 EM$ = STRING$(20,".")
252 EE$ = STRING$(02,".")
253 NT$ = "../../.."
254 GF$ = STRING$(09,".")
255 FF$ = "...........-.."
256 CT$ = STRING$(06,".")
257 ST$ = STRING$(02,".")
258 DT$ = "../../.."
260 NR$ = STRING$(05,".")
270 DR$ = "../../.."
280 CF$ = STRING$(20,".")
282 SF$ = STRING$(20,".")
284 SL$ = "............."
286 NP$ = STRING$(20,".")
288 BP$ = STRING$(03,".")
290 AP$ = STRING$(04,".")
292 DP$ = "../../.."
294 ND$ = STRING$(02,".")
296 FG$ = STRING$(02,".")
298 BG$ = STRING$(03,".")
300 AG$ = STRING$(04,".")
302 DG$ = "../../.."
400 ' -----------
402 ' Dados Fixos
404 ' -----------
410 CLS
411 PRINT
412 PRINT SPC(4) "Cadastramento de Funcionarios"
414 PRINT STRING$(38,"=")
415 PRINT
416 PRINT "Nome:"
417 PRINT "Endereco:"
420 PRINT
422 PRINT "CEP:"SPC(6)"Cidade:"
424 PRINT "Estado:"SPC(3)"Nascimento:"
426 PRINT "RG:"SPC(10)"CIC:"
428 PRINT "CTPS:"SPC(7)"serie:"SPC(3)"emissao:"
```

```
430 PRINT
432 PRINT "Registro:"SPC(6)"Em:"
434 PRINT "cargos:"
436 PRINT "secao:"
438 PRINT "salario:"
440 PRINT
442 PRINT "PIS:"
444 PRINT "banco:"SPC(4)"agencia:"SPC(5)"data:"
446 PRINT "dependentes:"
448 PRINT
450 PRINT "FGTS:"
452 PRINT "BANCO:"SPC(4)"agencia:"SPC(5)"data:";
500 ' ---------------
502 ' Dados Variaveis
504 ' ---------------
506 LOCATE  5, 4:PRINT NF$
508 LOCATE  9, 5:PRINT EF$
510 LOCATE  4, 7:PRINT EC#
512 LOCATE 17, 7:PRINT EM$
514 LOCATE  7, 8:PRINT EE$
516 LOCATE 21, 8:PRINT NT$
518 LOCATE  3, 9:PRINT GF$
520 LOCATE 17, 9:PRINT FF$
522 LOCATE  5,10:PRINT CT$
524 LOCATE 18,10:PRINT ST$
526 LOCATE 29,10:PRINT DT$
528 LOCATE  9,12:PRINT NR$
530 LOCATE 18,12:PRINT DR$
532 LOCATE  6,13:PRINT CF$
534 LOCATE  6,14:PRINT SF$
536 LOCATE  8,15:PRINT SL$
538 LOCATE  4,17:PRINT NP$
540 LOCATE  6,18:PRINT BP$
542 LOCATE 18,18:PRINT AP$
544 LOCATE 28,18:PRINT DP$
546 LOCATE 12,19:PRINT ND$
548 LOCATE  5,21:PRINT FG$
550 LOCATE  6,22:PRINT BG$
552 LOCATE 18,22:PRINT AG$
554 LOCATE 28,22:PRINT DG$
600 LOCATE 0,0
```

A separação da impressão dos dados fixos dos dados variáveis irá agilizar a digitação de dados, pois o programa retornará, normalmente, a linha 500 e não perderá tempo imprimindo novamente os dados fixos que já estão impressos na tela.

NÚMEROS DECIMAIS COM VÍRGULA

Os números fracionários são apresentados na notação internacional com o ponto decimal, o que poderá ocasionar certa confusão.

Recomenda-se substituir o ponto decimal pela vírgula decimal e nos casos de números grandes, inserir o ponto de separação milesimal.

Isso pode ser obtido por uma rotina do tipo:

```
100 INPUT "Numero = ";N
102 GOSUB 2400
104 PRINT A$
106 END
2400 ' ----------------------------------
2402 ' RTCONVPV = Converte Ponto em Virgula
2404 '-----------------------------------
2410 A$ = STR$(N)
2412 K = INSTR(A$,".")
2414 IF K = 0 THEN A$=A$+".00"
2416 K = INSTR(A$,".")
2418 A$ = LEFT$(A$,K-1) + "," + MID$(A$,K+1)
2420 K = INSTR(A$,",")
2422 IF LEN(LEFT$(A$,K-1))>4 THEN
     A$=LEFT$(A$,K-4)+"."+MID$(A$,K-3)
2424 K = INSTR(A$,".")
2426 IF K>0 THEN IF LEN(LEFT$(A$,K-1)) THEN
     A$=LEFT$(A$,K-4)+"."+MID$(A$,K-3)
2428 RETURN
```

6

manipulação de variáveis

Em um programa BASIC, as informações devem ser armazenadas em variáveis.

Para que o programador, na análise do programa, não seja levado a cometer erros grosseiros, confundindo o conteúdo das variáveis, por causa de uma interpretação errônea de seus nomes, sugere-se adotar alguns padrões.

A linguagem BASIC, por si só, apresenta algumas restrições, amplamente comentadas no livro Aprofundando-se no MSX. De forma resumida essas restrições são as seguintes:

- O nome da variável pode ser formado por até 252 caracteres, porém, somente os 2 primeiros são considerados no nome da variável.
- Variáveis que contenham números inteiros devem ser finalizadas com o símbolo % .
- Variáveis que contenham números reais de precisão simples devem ser finalizadas com o símbolo ! .
- Variáveis que contenham números reais de dupla precisão devem ser finalizadas com o símbolo # .

- Variáveis que contenham informações alfa-numéricas devem ser finalizadas com o símbolo $.

- Variáveis que contenham qualquer tipo de informação, quer numéricas quer alfa-numéricas, que sirvam para definir os campos de um registro de um arquivo randômico devem ser finalizadas com o símbolo $ e a quantidade de bytes utilizados por essas variáveis depende do tipo de informação conforme a tabela seguinte:

números inteiros	= 2 bytes
números reais em precisão simples	= 4 bytes
números reais em dupla precisão	= 8 bytes
informações alfa-numéricas	= 1 byte por caractere

É difícil tentar estabelecer um padrão para o nome das variáveis, mas analisando as restrições de outras linguagens de programação, podemos tentar estabelecer determinadas regras:

1 - Reservar as iniciais de I a M para as variáveis de controle.

2 - Quando houver várias variáveis semelhantes, habituar a fazer a diferenciação na primeira letra da variável. Assim, em vez de MES1 usar PMES.

3 - No desenvolvimento de um grande sistema, quando participam vários programadores, procurar elaborar uma tabela onde poderão ser relacionadas as variáveis que serão utilizadas nos diversos programas.

CONVERSÕES DE TIPOS

1 - número para string.

A conversão de números em strings podem ser realizadas por meio da função STR$, independentemente do tipo de variável numérica.

```
A$ = STR$(I%)
A$ = STR$(J!)
A$ = STR$(K)
A$ = STR$(L#)
```

2 - número para string de definição de campo.

A conversão de número em string de definição de campo de um registro de arquivo randômico, deve ser realizada por funções, de acordo com o tipo de variável numérica.

```
N$ = MKI$(I%)
N$ = MKS$(J!)
N$ = MKS$(K)
N$ = MKD$(L#)
```

3 - string para número.

A conversão de strings em números pode ser realizadas por meio da função VAL:

```
I% = VAL(A$)
J! = VAL (A$)
K  = VAL(A$)
L# = VAL(A$)
```

4 - string de definição de campo para número.

A conversão de strings de definição de campos de um registro de um arquivo randômico para números deve ser realizada por funções, de acordo com o tipo de variável numérica.

```
I% = CVI(N$)
J! = CVS(N$)
K  = CVS(N$)
L# = CVS(N$)
```

5 - string com algarismos hexadecimais para número.

```
N = VAL("&H"+A$)
```

MANIPULAÇÃO DE STRINGS

1 - TRANSFERÊNCIAS

As transferências de informações alfa-numéricas de uma variável para outra pode ser feita diretamente:

```
A$ = B$
```

2 - JUNÇÃO

A junção ou concatenação de strings pode ser feita por intermédio do símbolo + .

Exemplo:

```
100 A$ = "BOA"
102 B$ = "TARDE"
104 C$ = A$ + B$
106 PRINT C$
108 D$ = A$ + "NOITE"
110 PRINT D$
```

3 - PARTES DE STRINGS

Na situação em que se deseja somente uma parte da informação alfa-numérica, podemos utilizar uma das funções:

- LEFT$
- RIGHT$
- MID$

O emprego dessas funções é amplamente discutido no livro Linguagem Basic MSX.

Essas funções podem ser utilizadas em combinação com outras funções. Vejamos alguns exemplos:

Deseja-se imprimir somente o nome do logradouro, lendo esses nomes em um arquivo onde o nome do logradouro está embutido no endereço.

```
100 OPEN "CADASTRO.ARQ" AS #1 LEN=87
102 FIELD #1,30 AS N$,50 AS E$, 5 AS CE$,2 AS L$
104 MAX = LOF(1)/87
```

```
110 FOR REG = 1 TO MAX
112   LPRINT E$
114 NEXT REG
116 END
```

Este programa irá produzir uma listagem do tipo:

```
rua Joaquim Antunes, 450
av. Miruna, 34
rua General Osorio, 700
praca Gomes Carneiro, 1.234 - apto 12
```

Mas, desejamos uma listagem somente com o nome do logradouro, sem as palavras "rua", "av." etc.

```
Joaquim Antunes
Miruna
General Osorio
Gomes Carneiro
```

Isso pode ser obtido com um programa do tipo:

```
100 OPEN "CADASTRO.ARQ" AS #1 LEN=87
102 FIELD #1,30 AS N$,50 AS E$,5 AS CE$,2 AS F$
104 MAX = LOF(1)/87
106 FOR REG=1 TO MAX
108   I = INSTR(E$," ")
110   F = INSTR(E$," ") - I
112   LPRINT MID$(E$,I,F)
114 NEXT REG
116 END
```

VÍRGULA E PONTO DECIMAL

A notação empregada no Brasil para números fracionários é com vírgula decimal.

Ocorre que os computadores, por seguir uma padronização universal, utiliza o ponto decimal na notação de números.

A substituição da vírgula por ponto e vice-versa pode ser efetuada de forma mais ou menos simples convertendo-se o número em string e aplicando-se a função INSTR.

Para converter ponto para vírgula decimal:

```
300 A$ = STR$(X)
302 P = INSTR(A$,".") - 1
304 IF P >= 0 THEN A$ = LEFT$(A$,P)+","+
              RIGHT$(A$,LEN(A$)-P-1)
```

Para converter vírgula para ponto decimal:

```
300 P = INSTR(A$,",") - 1
302 IF IF P >= O THEN A$ = LEFT$(A$,P)+"."+
                RIGHT$(A$,LEN(A$)-P-1)
304 X = VAL(A$)
```

7

desenvolvimento de um sistema

Para desenvolver um programa ou um conjunto de programas, devemos proceder a Análise do Sistema.

Em que consiste essa análise ?

Analisar um sistema significa analisar os dados que serão manipulados pelos programas definindo-se os arquivos do sistema e, em seguida, analisar os arquivos definindo-se os programas do sistema.

Da análise do sistema, resultam, portanto, um conjunto de arquivos e de programas que constituem o Sistema. Em outras palavras, damos o nome de Sistema a um cojunto de arquivos e programas estruturados para resolver um determinado problema.

Por que um conjunto e não um único programa?

Várias razões nos levam a dividir um grande programa em vários outros menores. Algumas dessas razões podem ser as seguintes:

1 - O computador não dispõe de memória suficiente para conter um único programa.

2 - Durante o processamento de certas fases do problema será feita, por exemplo, uma ordenação dos dados para colocar os dados em certa ordem e para isso será empregada uma técnica de classificação cuja velocidade dependerá da quantidade de memória livre disponível.

3 - O sistema caracteriza-se por intenso acesso a arquivos em disco sendo acessado vários arquivos ao mesmo tempo e para permitir uma grande flexibilidade ao usuário, vários parâmetros dos arquivos poderão ser definidos na memória, e a eficiência das técnicas de acesso estará diretamente associada a quantidade de memória livre.

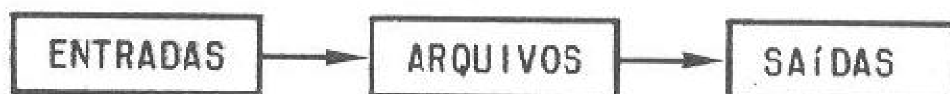
Um programador experimentado apresentará dezenas de outras razões segundo as quais um sistema deve, sempre que possível, ser dividido em programas pequenos.

Para definir os arquivos do sistema, o programador deve:

1 - Preparar uma relação contendo todas as informações que deverão ser fornecidas pelo sistema. Para ser prático, desenhar em um papel quadriculado, todos os relatórios que se deseja do sistema.

2 - Partindo do pressuposto que o computador não cria informações, analisar as informações acima e preparar uma relação de todas as informações que deverão ser fornecidas ao sistema. Agrupar essas informações em função da época em que elas são produzidas. Para ser prático, desenhar, em um papel quadriculado, a forma em que os dados serão digitados.

3 - Tomando como base o esquema seguinte:



Analise o fluxo dos dados desde a Entrada até a Saída e procure uma estrutura de arquivos onde os dados sejam manipulados o menos possível, lembrando que as leituras e gravações em disco consomem tempo.

Nessas análises, faça várias hipóteses, desde a configuração em que todas as informações ficam armazenadas em um único arquivo, até a configuração em que os dados ficam distribuidos em vários arquivos.

Após essas análises resultará a configuração final dos arquivos, tendo-se a definição de quais são os arquivos e quais as informações que serão armazenadas em cada um desses arquivos.

Para se definir os programas do sistema, o programador deve:

1 - Desenhar os arquivos em uma folha grande.

2 - Adicionar no desenho, os programas de emissao dos relatórios, ligando-os aos arquivos.

3 - Adicionar no desenho, os programas de digitação de dados.

4 - Prever as situações em que haverá a necessidade de corrigir os dados já arquivados e adicionar os programas que farão tais correções.

5 - Prever as situações em que haverá a necessidade de se eliminar algum registro do arquivo e adicionar os programas que farão tais eliminações.

Após isso, estará definido o sistema de arquivos e programas. Fazer a Especificação do Sistema. Para isso, o programador deve:

1 - Especificação dos Arquivos:

Para cada arquivo do sistema:

A - Definir o nome do arquivo;

B - Definir o tipo de acesso, se sequencial ou randômico;

C - Para cada campo do arquivo:

1 - Descrever o conteúdo;
2 - Definir o nome da variável de acesso.
3 - Limitar o tamanho da informação;

2 - Especificação dos Programas:

Para cada programa do sistema:

1 - Definir o nome do programa;
2 - Descrever o que ele faz;
3 - Desenhar telas de controle do programa;
4 - Desenhar as telas de digitação de dados;
5 - Desenhar a estrutura lógica dos programas em fluxogramas.

Aproveite a oportunidade para preparar uma justificativa, isto é, relacionar os motivos que conduziram os trabalhos de análise.

Como resultado da Análise do Sistema, teremos o seguinte material:

1 - Especificação dos programas.
2 - Especificação dos arquivos.
3 - Desenho dos relatórios impressos.
4 - Desenho das telas de digitação de dados.
5 - Desenho das telas de controle.
6 - Fluxograma dos programas.
7 - Fluxograma do sistema.
8 - Justificativas.

Guarde esse material para compor o Manual do Sistema.

EXEMPLO DE UMA ANÁLISE DE SISTEMA

Vamos efetuar a análise de sistema para a elaboração de um Sistema de Controle de Materiais do almoxarifado de uma empresa.

ANÁLISE DO SISTEMA DE ARQUIVOS

Vamos seguir, passo a passo, o roteiro acima apresentado, efetuando a análise do exemplo apresentado no capítulo 9.

1 - Informações que se espera do sistema:

Desenhar, em papel quadriculado, todos os relatórios que se deseja obter do sistema. Entende-se como relatório toda e qualquer informação que seja apresentada pelo computador em resposta a uma solicitação feita pelo usuário.

Esses desenhos estão apresentados no capítulo 9. Com base neles, podemos apresentar o seguinte:

INFORMACÃO	RELATÓRIO 1	2	3	4
- Nome da Firma	X	X	X	X
- Título do Relatório	X	X	X	X
- Numeração das Folhas	X	X	X	X
- Data da Emissão	X	X	X	X
- Código do Material	X			X
- Descrição do Material	X	X	X	X
- Saldo Atual em Estoque	X			X
- Data da Última Entrada	X			
- Quantidade de Itens no Cadastro	X			
- Data da Entrada		X		
- Data da Saída		X	X	
- Número do Documento (Histórico)		X		
- Quantidade de Entrada		X		X
- Quantidade de Saída		X	X	X
- Quantidade Anterior				X

Essas são todas as informações que se deseja obter do sistema.

O analista do sistema deve tomar extremo cuidado na definição dos relatórios, pois, como veremos a seguir, é a variedade de informações que define a complexidade do sistema. Porisso, o analista experiente deve até antever algum relatório que no futuro venha a ser necessário.

2 - Classificação das Informações.

Montada a relação, analisar bem o tipo das informações. Observar que existem informações fixas, que nunca variam, e outras que variam com uma certa frequência.

Separando umas das outras teremos:

- Informações Fixas:
  - Nome da Firma
  - Título do Relatório

- Informações que variam em função do relatório:
  - Numeração das Folhas
  - Data da Emissão

- Informações que variam de acordo com o estoque:
  - Código do Material
  - Descrição do Material
  - Saldo Atual em Estoque
  - Data da Última Entrada
  - Quantidade de Itens no Cadastro
  - Data da Entrada
  - Data da Saída
  - Número do Documento (Histórico)
  - Quantidade de Entrada
  - Quantidade de Saída
  - Quantidade Anterior

Analisar as informações variáveis e imaginar as situações em que elas serão produzidas.

No caso em questão, tais situações são as seguintes:

- Época de Cadastramento, quando a empresa passou a adquirir um novo material.

Para um certo material, essa época existe uma única vez.

- Época de aquisição, quando é realizada a entrada no almoxarifado.

Para um certo material essa época pode ocorrer com uma grande frequência.

- Época de consumo, quando é realizada a saída do almoxarifado.

Para um certo material essa época pode ocorrer com uma grande frequência.

Classificando as informações variáveis de acordo com as épocas, teremos o seguinte:

- Época de Cadastramento:
  - Código do Material
  - Descrição do Material

- Época de Aquisição:
  - Data da Entrada
  - Número do Documento (Histórico)
  - Quantidade de Entrada

- Época de Consumo:
  - Data da Saída
  - Número do Documento (Histórico)
  - Quantidade de Saída

Da relação anterior ficam sobrando algumas informações que não estão relacionadas diretamente às épocas acima:

- Saldo Atual em Estoque
- Quantidade de Itens no Cadastro
- Quantidade Anterior

Vejamos como será feita a digitação das informações. Para a digitação dos dados é necessário desenhar uma tela de digitação onde as informações sejam distribuídas no espaço de 24 x 40 (ou 80 se o vídeo tiver 80 colunas) com uma estética que facilite a verificação visual daquilo que se digita.

Imaginando que futuramente serão necessários relatórios com os materiais agrupados por tipos como "material de escritório", "de limpeza", "do refeitório", etc, vamos incluir a informação TIPO.

Imaginando, também, que no futuro seja necessário um controle do estoque baseado em uma quantidade mínima, vamos incluir a informação Quantidade Mínima.

O desenho da tela de Cadastramento de Material é apresentado no capítulo 9.

Para a época de aquisição de materiais será necessária uma tela de digitação quando serão digitadas:

- Data da Entrada
- Número do Documento
- Quantidade da Entrada

Imaginando a situação em que o usuário irá digitar essas informações, a tela deverá apresentar algumas informações para orientá-lo.

Por exemplo, após a digitação do código do material, seria interessante que o sistema respondesse com a descrição correspondente para que o digitador possa ter a certeza de que os dados que serão digitados referem-se ao meterial desejado.

O desenho da tela de Registro de Entrada é apresentado no capítulo 9.

Efetuando uma análise similar, podemos desenhar a tela de Registro de Saída apresentada no capítulo 9.

3 - ANÁLISE DO ARQUIVO

As informações digitadas, independentemente das épocas em que são produzidas, devem ser gravadas em um arquivo, para ali permanecerem até o dia em que serão consultadas para a emissão dos relatórios.

A idéia mais imediata seria armazenar todas as informações em um único arquivo.

Um registro desse arquivo teria o seguinte tamanho:

- Código do Material	4
- Descrição	25
- Data da Entrada	6
- Número do Documento de Entrada	10
- Quantidade de Entrada	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5
TOTAL =	71 bytes

Observe que foram definidas apenas uma Entrada e uma Saída. Ocorre que existem materiais que são consumidos a todo instante, gerando inúmeras Saídas ao longo do mês.

Uma das alternativas seria prever vários campos no próprio registro:

- Código do Material	4
- Descrição	25
- Data da Entrada	6

- Número do Documento de Entrada	10
- Quantidade de Entrada	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5

TOTAL = 155 bytes

Nos arquivos, tanto do tipo sequencial como no randômico, a quantidade de campos de um registro deve, obrigatoriamente, ser fixo, não podendo se ter alguns registros com 8 campos e outros com 20. Todos o registros do arquivo devem ter a mesma quantidade de campos.

Na presente alternativa, deixando-se um grande espaço para poder registrar as inúmeras saídas de determinados materiais de grande consumo, iremos incorrer no desperdício de espaço em disco pois haverá materiais que nem serão movimentados no mês.

Outra alternativa seria a de separar as informações em 2 arquivos:

- Arquivo 1:

- Codigo do Material	4
- Descrição	25
- Data da Entrada	6
- Número do Documento de Entrada	10
- Quantidade de Entrada	5

- Arquivo 2:

- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5

No primeiro arquivo serão registrados os movimentos de Entrada e no outro serão registrados os movimentos de saída. Para cada Saída haverá um registro correspondente.

A vantagem está na economia de espaço em disco, pois só será criado um novo registro se houver movimento. No caso do material que é muito consumido e que gera muitos movimentos de saída, haverá um registro para cada saída, enquanto que aquele material que não foi movimentado não terá nenhum registro no arquivo 2.

Observe que os arquivos, do jeito que foram definidos, não permitem que seja identificada qual saída coresponde a qual material. É preciso incluir uma referência comum. No caso, escolhe-se como referência comum o Código do Material.

O arquivo 2 fica sendo constituído, então, pelos seguintes campos:

- Código do Material	4
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5

Da mesma forma em que um certo material pode ocasionar muitos movimentos de Saída, existe a possibilidade de um material dar muitas Entradas ao longo de um mês. Nessas situações iremos deparar com o mesmo problema de desperdício de espaço em disco.

Uma alternativa de se contornar esse problema seria a de se colocar as informações de Entrada em um arquivo separado.

Nessa alternativa, o nosso sistema será composto por 3 arquivos:

- Arquivo 1:

- Código do Material	4
- Descrição	25

- Arquivo 2:

- Código do Material	4
- Data da Entrada	6
- Número do Documento de Entrada	10
- Quantidade de Entrada	5

- Arquivo 3:

- Código do Material	4
- Data da Saída	6
- Número do Documento de Saída	10
- Quantidade de Saída	5

Observe que a aparência dos arquivos 2 e 3 são bem semelhantes. Se você se lembrar do que foi discutido no capítulo 4 sobre "buffer" verá que o trabalho com muitos arquivos poderá complicar um pouco a operação do sistema.

Então é melhor juntar os arquivos 2 e 3 em um único arquivo, acrescentando, porém, uma informação para poder diferenciar os registros de Entrada e de Saída.

Resultam, então, os seguintes campos para o arquivo de Movimento:

- Data do Movimento	6
- Histórico	10
- Código do Material movimentado	4
- Quantidade Movimentada	5
- Tipo do Movimento (Entrada ou Saída)	1

Vamos aproveitar para incluir nesse arquivo, algumas informações necessárias no sistema e que não seriam bem inseridas no arquivo de movimento. Essas informações são as seguintes:

- Tipo do Material
- Saldo do Período Anterior
- Saldo Atual do Estoque
- Quantidade Mínima no Estoque
- Data da Última Entrada
- Data da Última Saída

Resultam, então, os seguintes campos para o Cadastro de Materiais:

- Código do Material
- Descrição do Material
- Tipo do Material
- Saldo do Período Anterior

- Saldo Atual do Estoque
- Quantidade Mínima no Estoque
- Data da última Entrada
- Data da última Saída

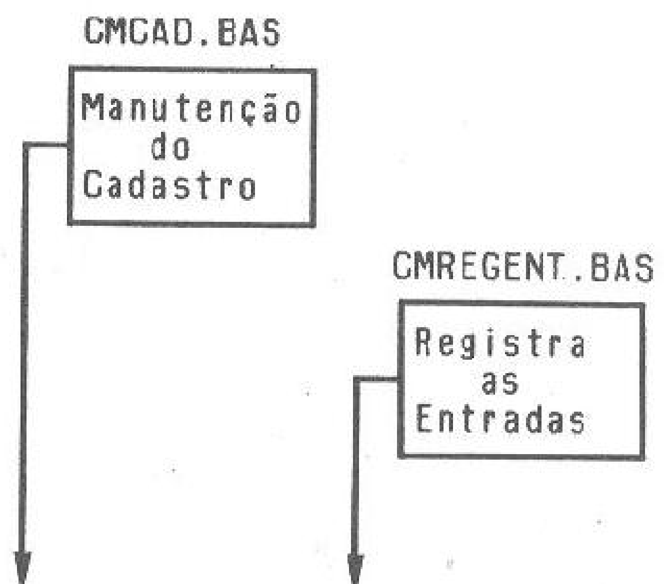
Para podermos ter um controle rígido sobre o sistema, vamos criar um outro arquivo, onde possamos armazenar algumas informações que não estejam relacionadas diretamente com os materiais e nem com os movimentos.

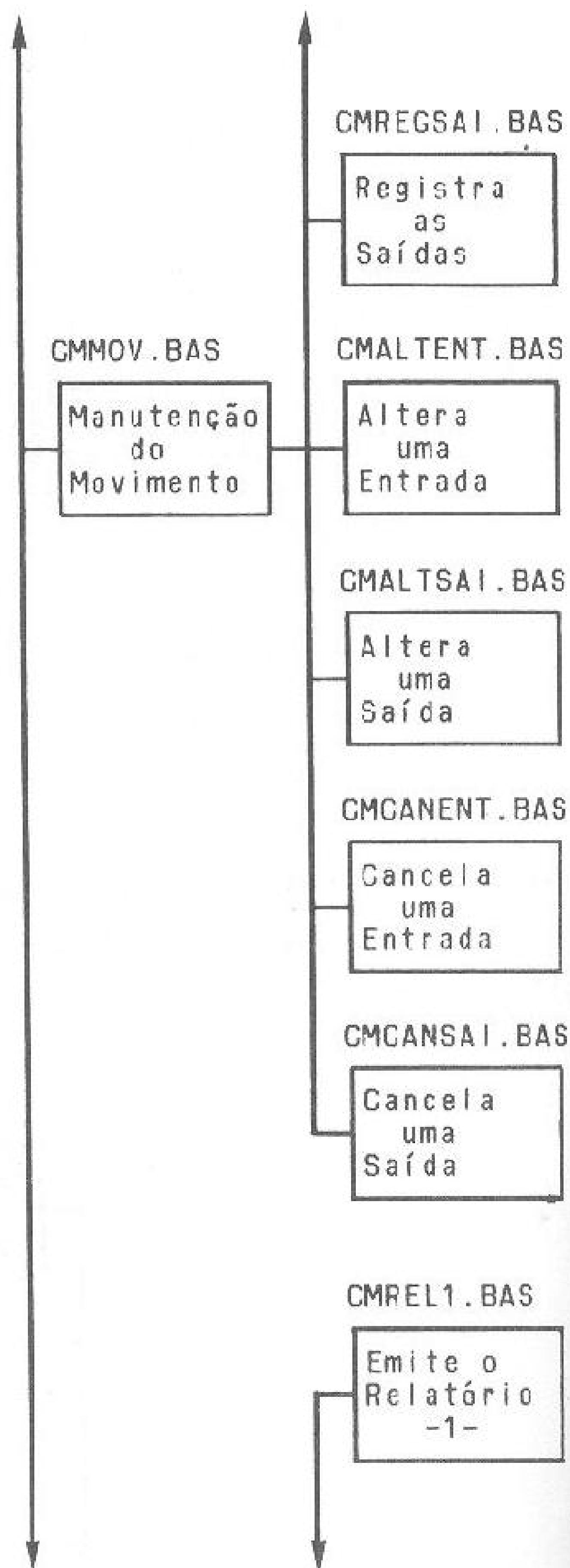
Como essas informações controlam o sistema, o arquivo será um arquivo de Parâmetros. As informações que fazem parte desse arquivo são as seguintes:

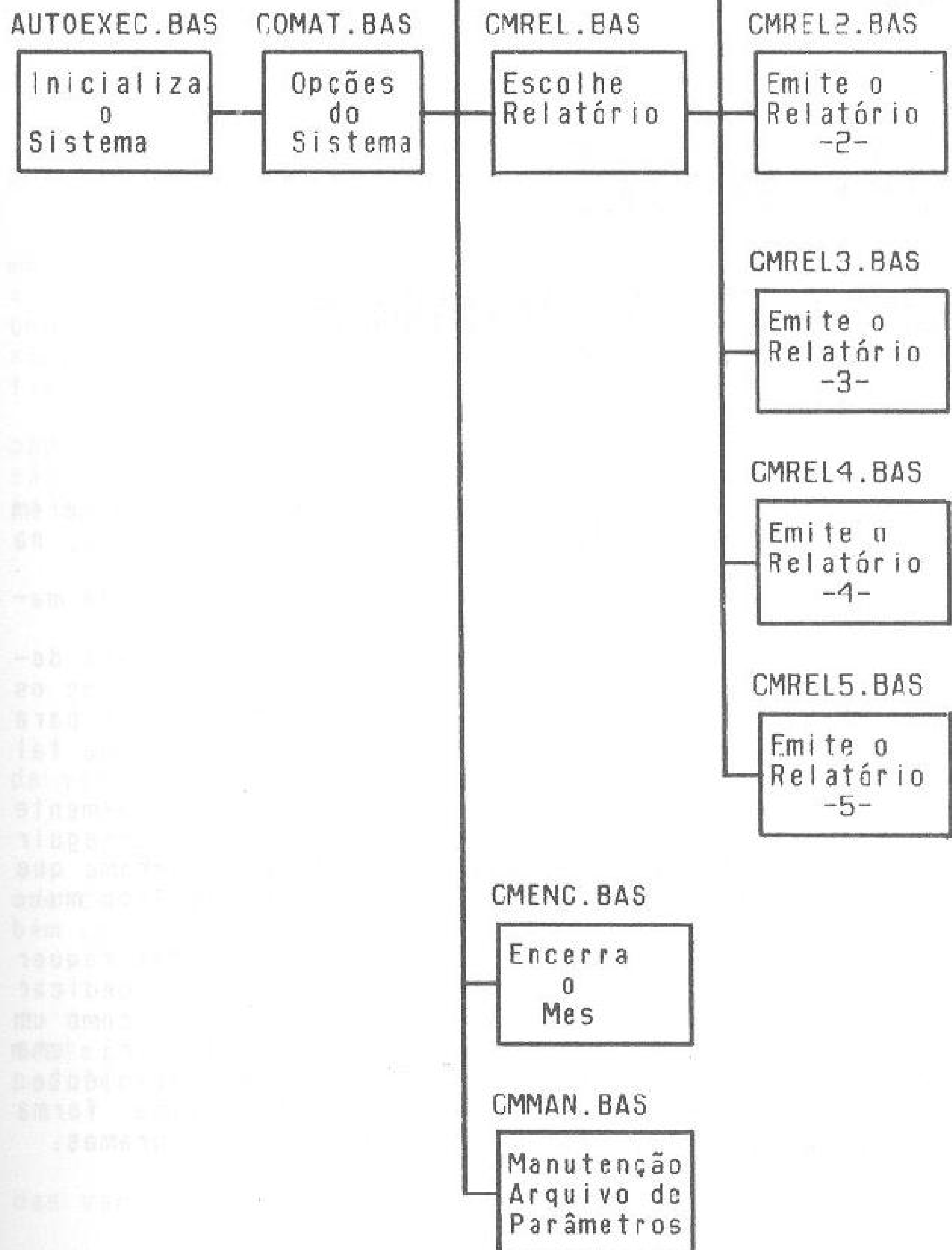
- Nome do Sistema
- Número do Mês em Andamento
- Nome do Mês em Andamento
- Quantidade de Registros no Cadastro de Materiais
- Quantidade de Registros no Arquivo de Movimento
- Última Data Acessada

ANÁLISE DO SISTEMA DE PROGRAMAS

Seguindo o roteiro apresentado no início deste capítulo, resultará o seguinte desenho do sistema:




CMREGSAI.BAS
Registra as Saídas
CMMOV.BAS
Manutenção do Movimento
CMALTENT.BAS
Altera uma Entrada
CMALTSAI.BAS
Altera uma Saída
CMCANENT.BAS
Cancela uma Entrada
CMCANSAI.BAS
Cancela uma Saída
CMREL1.BAS
Emite o Relatório -1-


AUTOEXEC.BAS
Inicializa o Sistema
COMAT.BAS
Opções do Sistema
CMREL.BAS
Escolhe Relatório
CMREL2.BAS
Emite o Relatório -2-
CMREL3.BAS
Emite o Relatório -3-
CMREL4.BAS
Emite o Relatório -4-
CMREL5.BAS
Emite o Relatório -5-
CMENC.BAS
Encerra o Mes
CMMAN.BAS
Manutenção Arquivo de Parâmetros

8

eficiência dos programas

Os problemas que são encaminhados para serem solucionados com o auxílio de computadores, o são, na maioria das vezes, para se ganhar tempo.

Isso significa que os programas, antes de mais nada, têm o compromisso de serem velozes.

A velocidade de execução de um programa depende, infelizmente, de vários fatores e nem todos os programadores possuem o conhecimento necessário para permitir que os programas sejam estruturados de tal forma a assegurar o menor tempo na execução.

Como veremos, "programar" não é simplesmente estabelecer um simples roteiro lógico. Para conseguir fazer a afirmação "Eu consegui fazer um programa que resolve tal problema" não é necessário um esforço muito grande.

A programação é uma arte e como tal requer uma dedicação especial. O programador que se dedicar firmemente em conhecer profundamente a maneira como um programa é executado pelo computador descobrirá uma série de macêtes que tornam o programa mais eficiente.

Vejamos alguns fatores que de alguma forma estejam relacionados com a velocidade dos programas:

1 - Ocupação da memória.
2 - Estrutura dos programas (como foram divididos e estão relacionados).
3 - Estrutura dos arquivos (como foram divididos e estão relacionados).

4 - Estrutura interna dos programas (como o programa está estruturado, onde estão as rotinas mais executadas, algorítimo empre gado).
5 - Programa Fonte e Programa Objeto.
6 - Compilação de Programas.

OCUPAÇÃO DA MEMÓRIA

Se fosse possível tirar uma fotografia da memória durante a execução de um programa, veríamos que a RAM é dividida em várias partes distintas, onde cada uma dessas partes possue uma atribuição própria e desempenha uma determinada função específica para permitir que um programa seja executado corretamente.

Vejamos, em detalhes, quais são essas partes, onde elas residem, a função de cada uma e como elas são estruturadas para que o desempenho do programa seja o mais eficiente possível.

Tais partes são as seguintes:

A - **Área do Programa**.
É a **região** em que o programa fica armazenado.

B - **Variáveis.**
É a região onde ficam armazenadas as tabelas de variáveis.

C - Matrizes.
É a região onde ficam armazenados os valores que correspondem as variáveis na forma de matriz, também conhecidas como Vetores ou Arrays.

D - Buffer.
É a região onde são armazenados, provisóriamente, os dados que transitam entre o programa e os periféricos conectados ao computador.

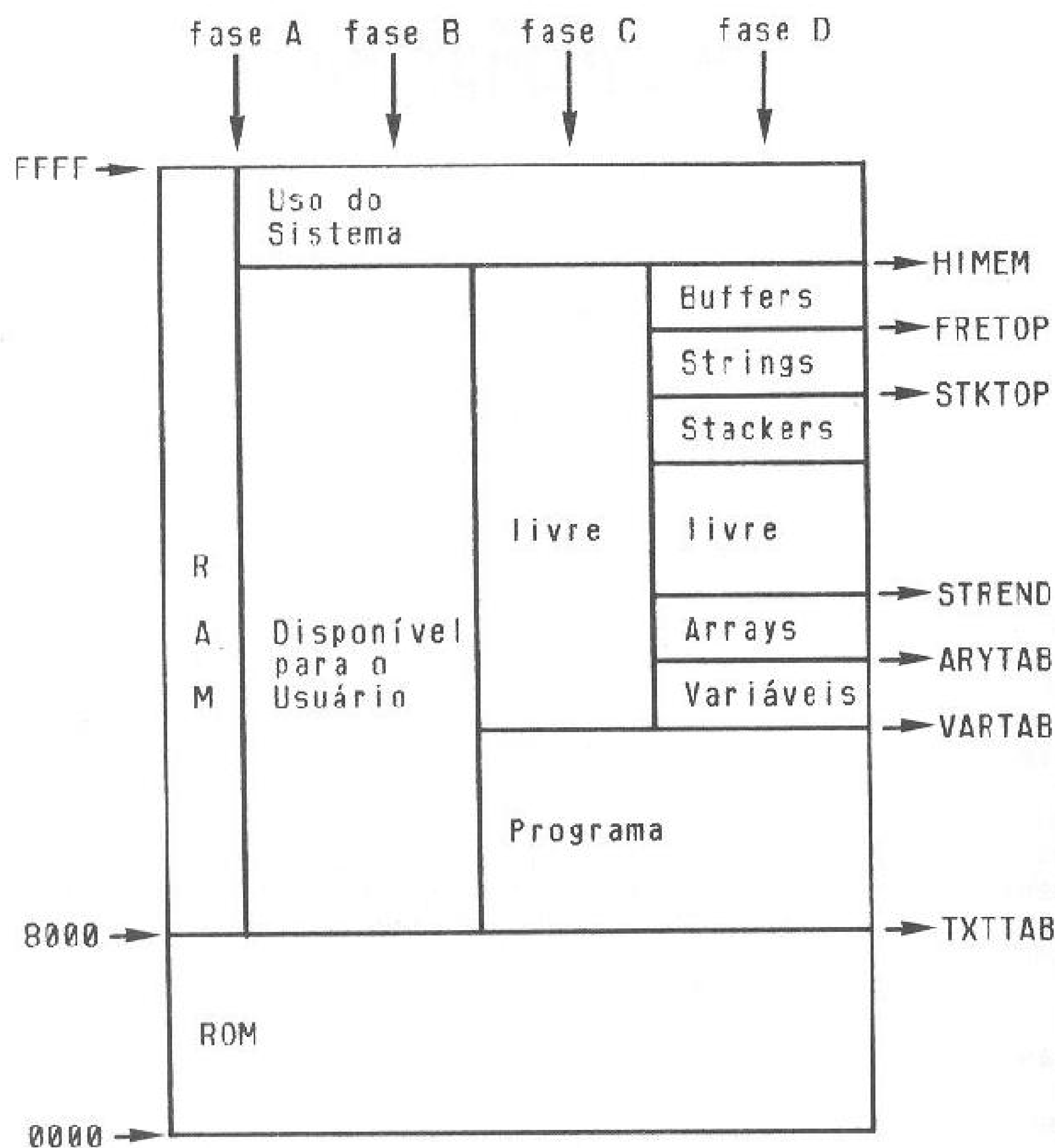
E - Strings.
É a região onde são armazenados os conteúdos das variáveis Strings do programa.

F - Stackers.
É a região onde são armazenadas as tabelas de endereços de controle do programa, também conhecidas como Stackers, pois tais tabelas são montadas na forma de pilhas.

G - Área Livre.

É a região da memória que sobra depois que foram distribuídos todos os espaços necessários para cada uma das partes.

Para agilizar o acesso as informacões, as áreas são agrupadas no início e no fim da RAM. Uma fotografia da RAM apresentaria algo do tipo:



As fases indicam o estado de utilização do computador e correspondem a:

A - O computador está desligado.
B - O computador está ligado, porém está sem nenhum programa na memória.
C - Um programa foi carregado na memória, mas ainda não foi executado.
D - O programa está em execução ou acabou de ser executado.

O apêndice II apresenta os mapas de ocupação da memória de alguns microcomputadores.

Vejamos, em detalhes, o conteúdo e a forma de cada uma das áreas:

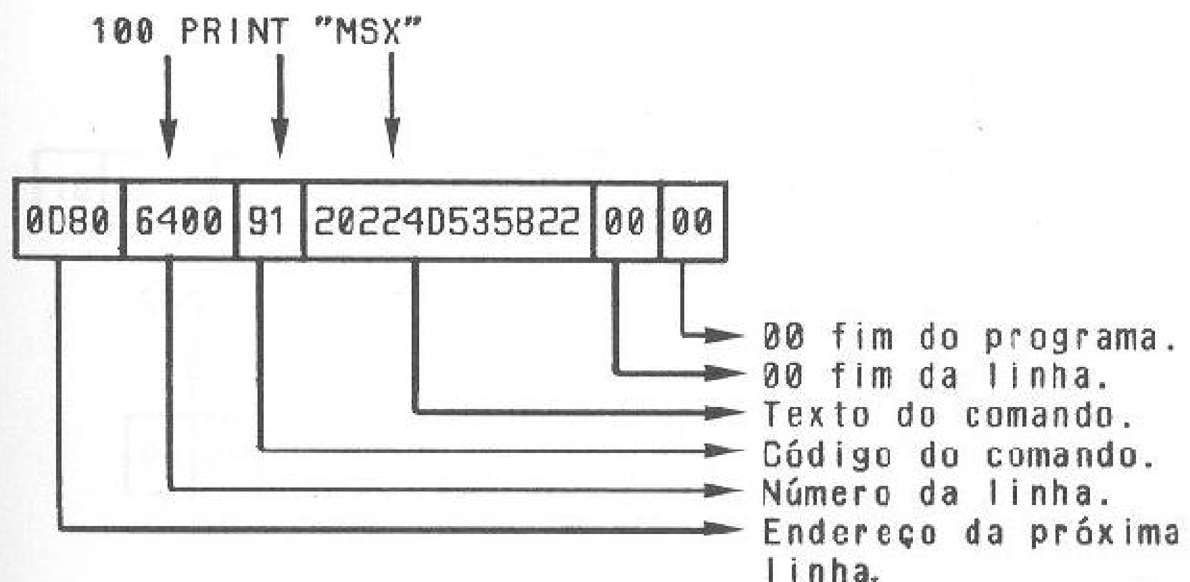
A - Área do Programa.

É a região em que o programa fica armazenado, na sequência em que as instruções foram digitadas.

Uma linha do programa fica armazenada na seguinte forma:

EEEE	LLLL	CC	texto	00

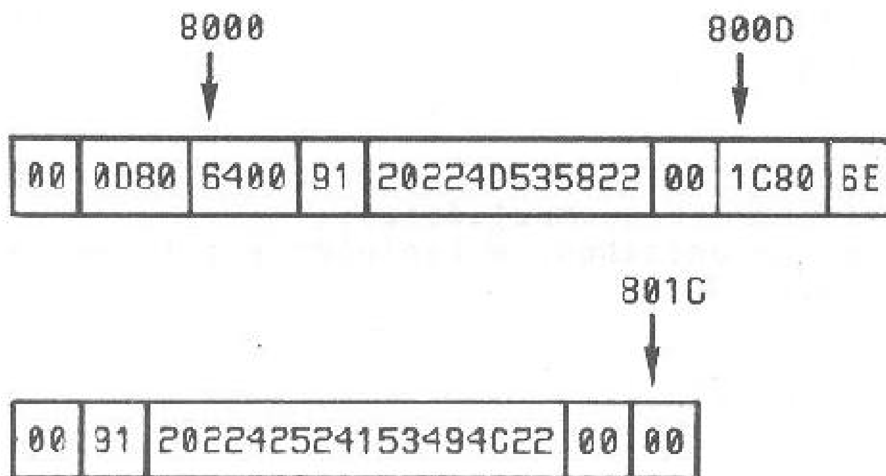
Exemplo:



A próxima linha do programa ficará em seguida, de modo que um programa com as instruções:

```
100 PRINT "MSX"
110 PRINT "BRASIL"
```

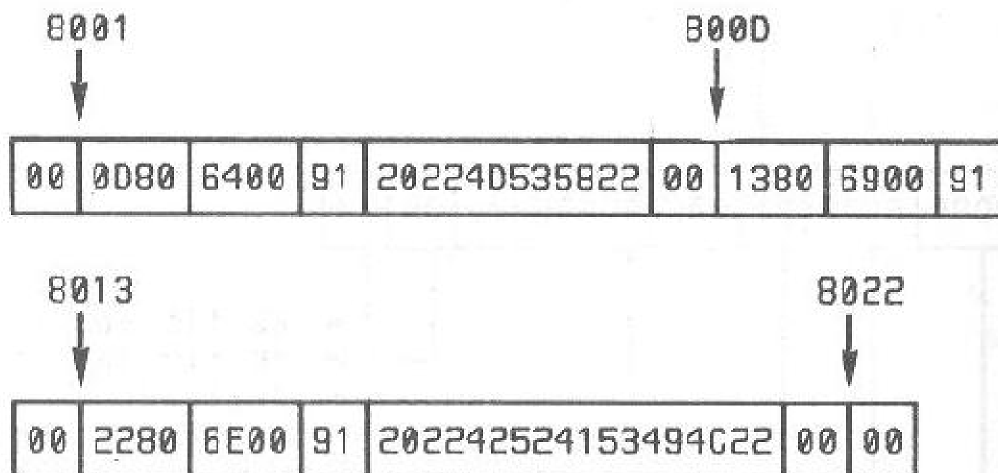
ficará com o seguinte aspecto, na memória:



Adicionando-se uma nova linha ao programa,

```
100 PRINT "MSX"
105 PRINT
110 PRINT "BRASIL"
```

o programa ficará com o seguinte aspecto na memória:



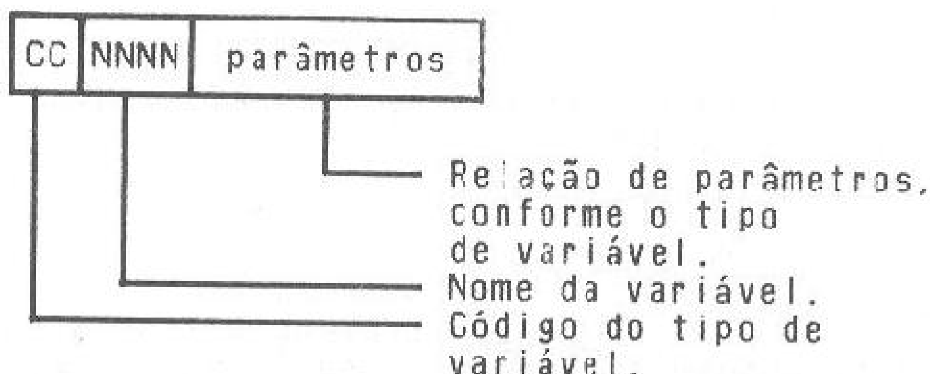
Como se vê, a cada nova linha que inserimos no programa, todas as linhas, que se situam após a linha inserida são deslocadas e os endereços de todas as linhas deslocadas são recalculados.

A área do programa tem início no endereço TXTTAB e termina no endereço VARTAB.

B - Variáveis.

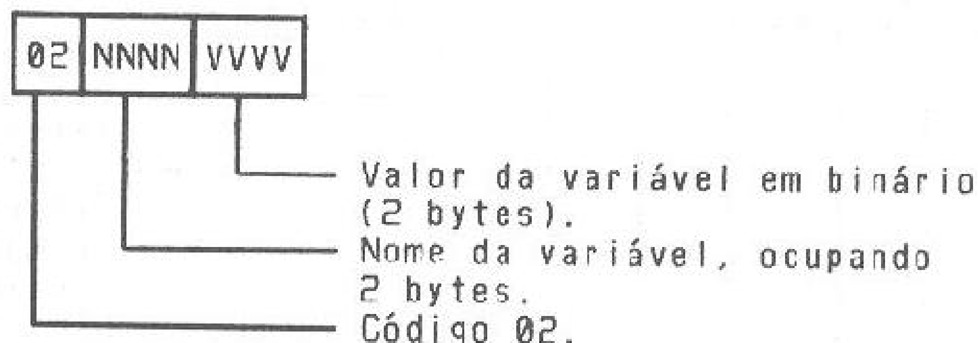
É a região onde ficam armazenadas as tabelas de variáveis.

A tabela de variáveis possue uma sintaxe genérica do tipo:

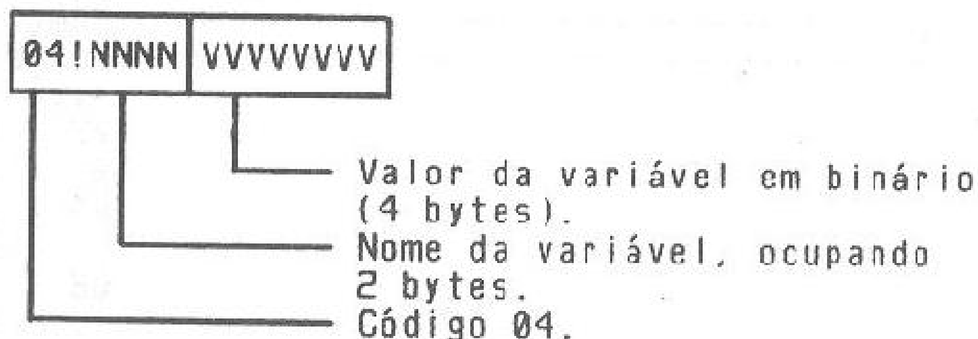


Para os diversos tipos de variáveis aceitas pelo **BASIC**, a tabela pode assumir uma das formas seguintes:

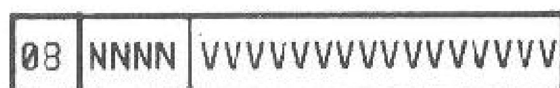
- Variável inteira:

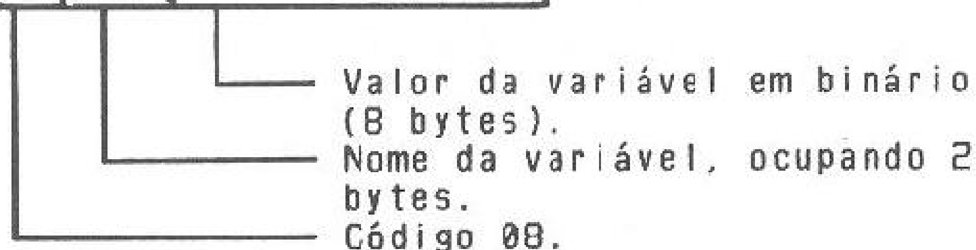


- Variável em ponto flutuante:



- Variável em precisão dupla:





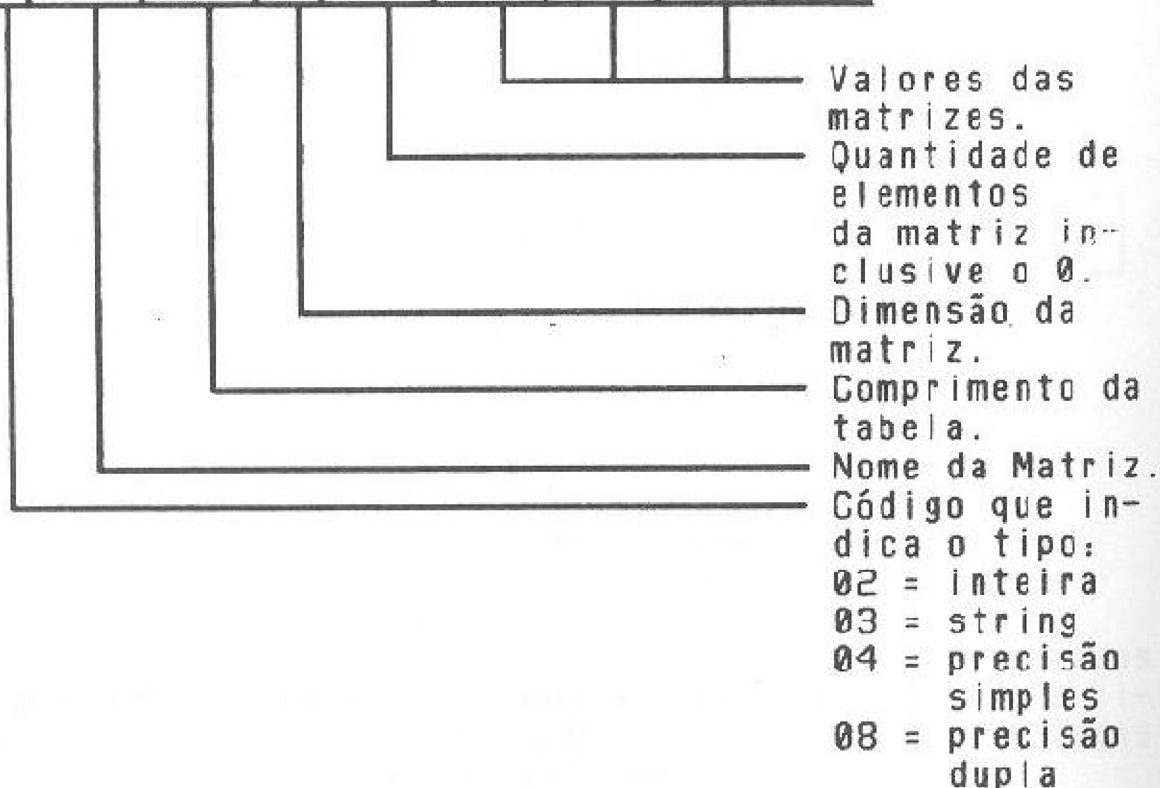
A área tem início no endereço VARTAB e termina no endereço ARYTAB.

C - Matrizes.

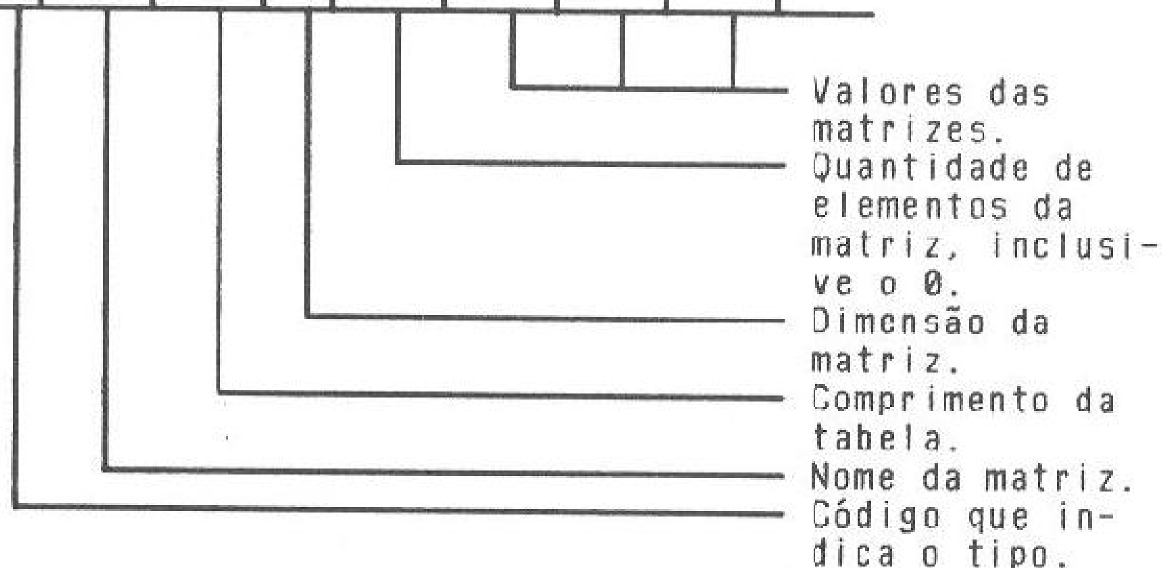
É a região onde ficam armazenados os valores que correspondem às variáveis na forma de matriz, também conhecidas como Vetores e Arrays.

A tabela de variáveis possue uma sintaxe genérica do tipo:

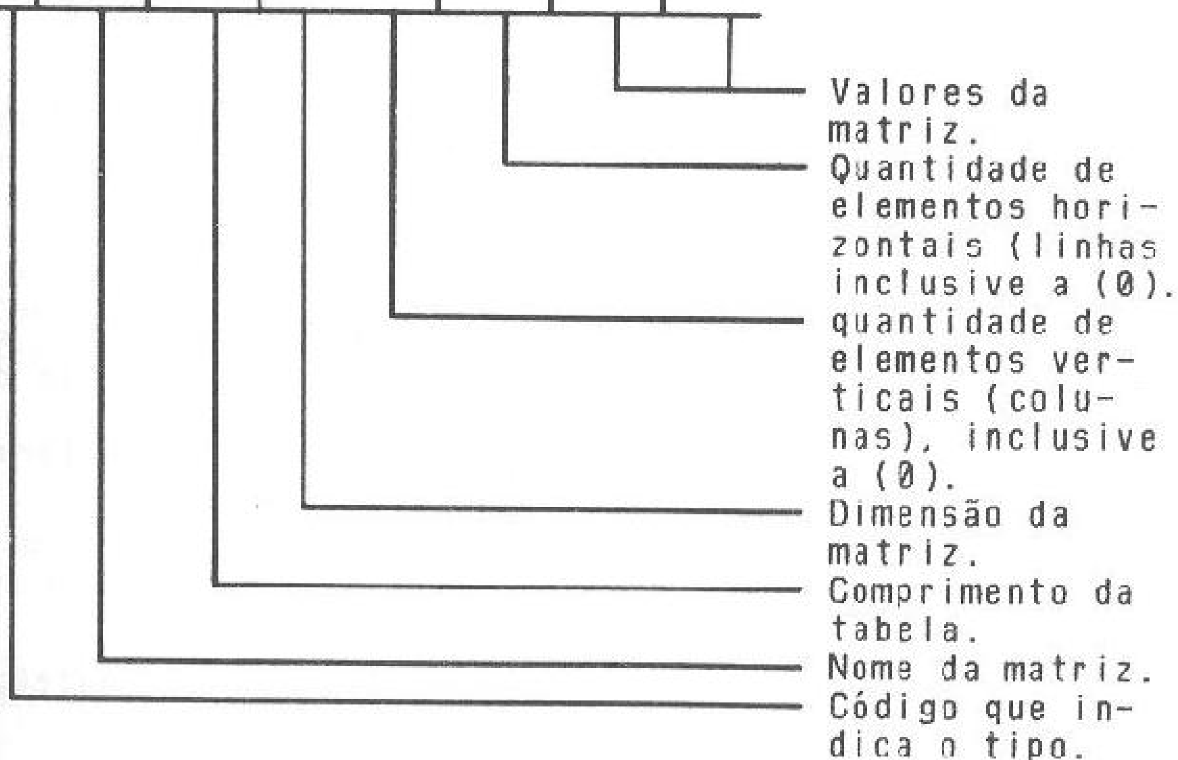




- Para matriz de 1 dimensão:



- Para matrizes de 2 dimensões:



Genericamente, podemos estabelecer a seguinte fórmula, para calcular o espaço necessário para uso de matrizes:

```
DIM variável(N)     ---   8+(N+1)*T
DIM variável(M,N)   ---   10+(M+1)*(N+1)*T
     etc.
```

onde T = 2 para variável inteira.
3 para variável string.
4 para variável de precisão simples.
8 para variável de precisão dupla.

A área tem início no endereço ARYTAB e termina no endereço STREND.

D - Buffer.

É a região onde são armazenados, provisoriamente, os dados que transitam entre o programa e os periféricos conectados ao computador.

Na área dos buffers, são reservados 267 bytes para cada arquivo que se abre.

A finalidade desses bytes é a seguinte:

- File Control Block		9
- Pointer		2
- Dados		256
	TOTAL =	267 bytes.

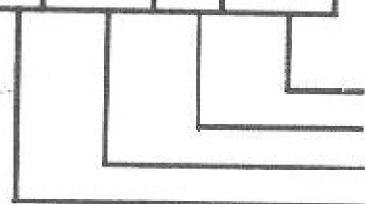
Inicialmente são reservados 536 bytes, o suficiente para caber 2 buffers, o que equivale ao comando MAXFILES = 2.

A área tem início no endereço FILTAB e termina no endereço HIMEM-1.

E - Strings.

É a região onde são armazenados os conteúdos das variáveis Strings do programa.





A área tem início no endereço STKTOP+1 e termina no endereço FRETOP.

O conteúdo da variável string, quando definida diretamente no programa, com uma instrução do tipo: A$="MSX", é a própria instrução, de modo que o endereço que o MSX armazena nos quinto e sexto bytes da tabela de definição da variável A$ apontará para o endereço dentro da instrução.

O conteúdo da variável string, quando redefinida, com uma instrução do tipo: A$=A$+"MSX", resultando em uma string diferente daquela com que a variável foi inicialmente definida, é armazenada na área de string. Nestes casos, o espaço ocupado é exatamente o comprimento da string.

Inicialmente o MSX reserva um espaço correspondente a 200 bytes, para evitar de ficar deslocando os Stackers a cada novo byte necessário na área de strings.

F - Stackers.

É a região onde são armazenadas as tabelas de endereços de controle do programa, também conhecidas como Stackers, pois tais tabelas são montadas na forma de pilhas.

Um desses enderecos é, por exemplo, o endereço de retorno de um GOSUB. Quando encontrar um comando RETURN, o sistema operacional consultará o STACK para saber onde ele estava antes de desviar para a rotina.

G - Área Livre.

É a região da memória que sobra depois que foram distribuídos todos os espaços necessários para cada uma das partes.

Como se vê, o espaço que um programa precisa para ser executado é bem maior que o tamanho do programa propriamente dito.

ESTRUTURA DOS PROGRAMAS

Para planejarmos o espaço que um programa ocupa na memória, devemos considerar, além do espaço acupado pelo programa, os seguintes espaços ocupados pelas variáveis, buffers e stackers:

- variáveis:

inteiras	5 bytes por variável
precisão simples	7 bytes por variável
precisão dupla	11 bytes por variável
strings	6 bytes por variável

- matrizes de uma única dimensão: (N)

inteiras	8+(N+1)*2
precisão simples	8+(N+1)*4
precisão dupla	8+(N+1)*8
strings	8+(N+1)*3

- matrizes de duas dimensões: (M,N)

inteiras	10+(M+1)*(N+1)*2
precisão simples	10+(M+1)*(N+1)*4
precisão dupla	10+(M+1)*(N+1)*8
strings	10+(M+1)*(N+1)*3

- matrizes de três dimensões: (L,M,N)

inteiras	12+(L+1)*(M+1)*(N+1)*2
precisão simples	12+(L+1)*(M+1)*(N+1)*4
precisão dupla	12+(L+1)*(M+1)*(N+1)*8
strings	12+(L+1)*(M+1)*(N+1)*3

- buffers: (MAXFILES=X) 2+X*267

- strings: somatória dos comprimentos das strings definidas pelo programa. Inicialmente é reservado um espaço de 200 bytes.

- stackers: 2 bytes para cada GOSUB.

A soma de todas essas áreas necessárias para a execução do programa não deve ser superior ao espaço disponível para o programa, que é a área que vai desde TXTTAB até HIMEM.

Num computador MSX sem disk drive, esse espaço é de 28.815 bytes, enquanto que em um computador MSX com disk drive, esse espaço é reduzido de 4 ou 5 Kbytes, aproximadamente.

Num computador IBM-PC, esse espaço é de 62.465 bytes.

Num computador APPLE com placa CP/M e 64 Kbytes de memória, esse espaço é de 26.483 bytes.

O que fazer se o espaço necessário for maior que esse limite?

Uma das alternativas é estudar a listagem do **programa** e procurar reduzir as áreas ocupadas pelas variáveis, o que pode ser feito por uma das seguintes maneiras:

1 - Diminuir a quantidade de variáveis.

Por exemplo, se no programa existem vários FOR...NEXT, procurar fazer com que tais comandos usem sempre a mesma variável de controle. Usar FOR N=1 TO 10 em um lugar e usar FOR I=1 TO 20 em outro lugar, isto é, a variável N para um caso e a variável I para outro caso, pode ser que facilite a interpretação do programa, mas obriga a um consumo adicional de 11 bytes, talvez sem necessidade.

2 - Usar o tipo de variável que ocupe menos espaço.

No mesmo exemplo anterior, se o FOR...NEXT for controlado por um STEP de número inteiro, não é necessário usar uma variável de precisão dupla. Usando, por exemplo, FOR N%=1 TO 10, você economizará 6 bytes. Lembre-se que o MSX assume como sendo de precisão dupla, todas as variáveis cujo tipo não esteja especificado com os símbolos %, ! ou # ou com os comandos DEFINT, DEFSGN ou DEFDBL.

Da mesma forma, certos tipos de variáveis como a IDADE deve sempre ser definida como uma inteira. Tal recomendação é muito importante, principalmente no caso em que a variável estiver na forma de uma matriz.

O comando DIM IDADE(200) reservará um espaço de 1.616 bytes, enquanto que o comando DIM IDADE%(200) reservará apenas 410 bytes, economizando, portanto, 1.206 bytes.

Outra alternativa seria dividir o programa em 2 outros programas menores, fazendo com que um deles chame o outro, o que apresenta uma série de vantagens, que podemos enumerar:

1 - Programas pequenos são mais rápidos para serem transferidos.

A transferência de um programa de um periférico onde ele está armazenado, por exemplo, fita cassete ou disco magnético, é feita byte a byte, de modo que programas mais longos consomem um tempo maior para serem transferidos.

Durante a confecção de um programa, é grande a frequência de SAVE e LOAD que efetuamos. Programas curtos serão transferidos em tempos menores, diminuindo sensivelmente o tempo que o programador terá que esperar para a execução de tais comandos.

Da mesma forma, durante a fase de testes do programa, quando imprime-se a listagem do programa com certa frequência, o tempo economizado nas impressões é bastante considerável.

2 - Programas pequenos são mais práticos de serem manipulados.

Como dito acima, a transferência de programas entre os periféricos é feita byte a byte. Durante a transferência podem ocorrer fatos inesperados, tais como uma oscilação na tensão de alimentação (voltagem) o que pode provocar uma perda de alguns bytes. O modo tecnicamente correto para se evitar tais fatos seria dotar o computador de um sistema de tensão garantida, adquirindo aparelhos conhecidos como "No-breaks", que pelo seu alto custo só são acessíveis a poucos privilegiados. Ao programador menos privilegiado só resta mesmo a alternativa de tentar fazer com que o tempo de transferência seja o menor possível, fazendo com que o programa seja o mais curto possível.

A divisão de um programa em 2 ou vários outros programas menores é perfeitamente válida, pois um programa, independentemente do seu tamanho é executa-

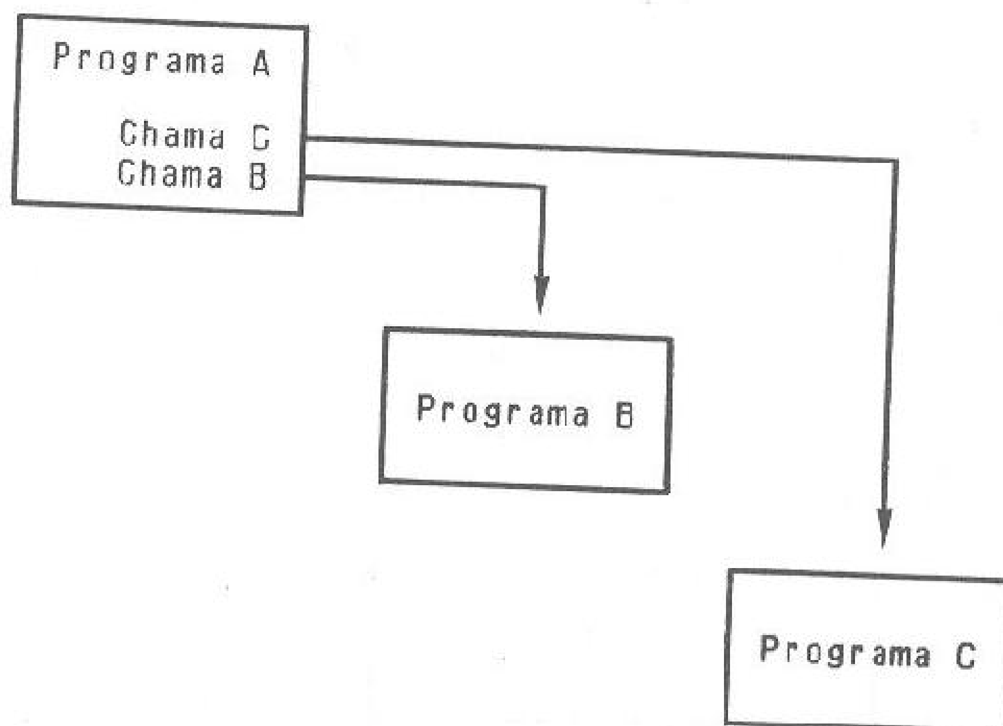
do pelo computador instrução por instrução, e mesmo em programas longos, em um dado instante, o computador estará executando uma determinada instrução, apenas aquela única instrução. Então, qual é a vantagem de se ter carregado na memória o programa todo com todas as opções possíveis se o computador irá executar uma única instrução de cada vez?

No futuro, os computadores serão dotados da capacidade de executar milhares de instruções simultaneamente. **Até lá ....**

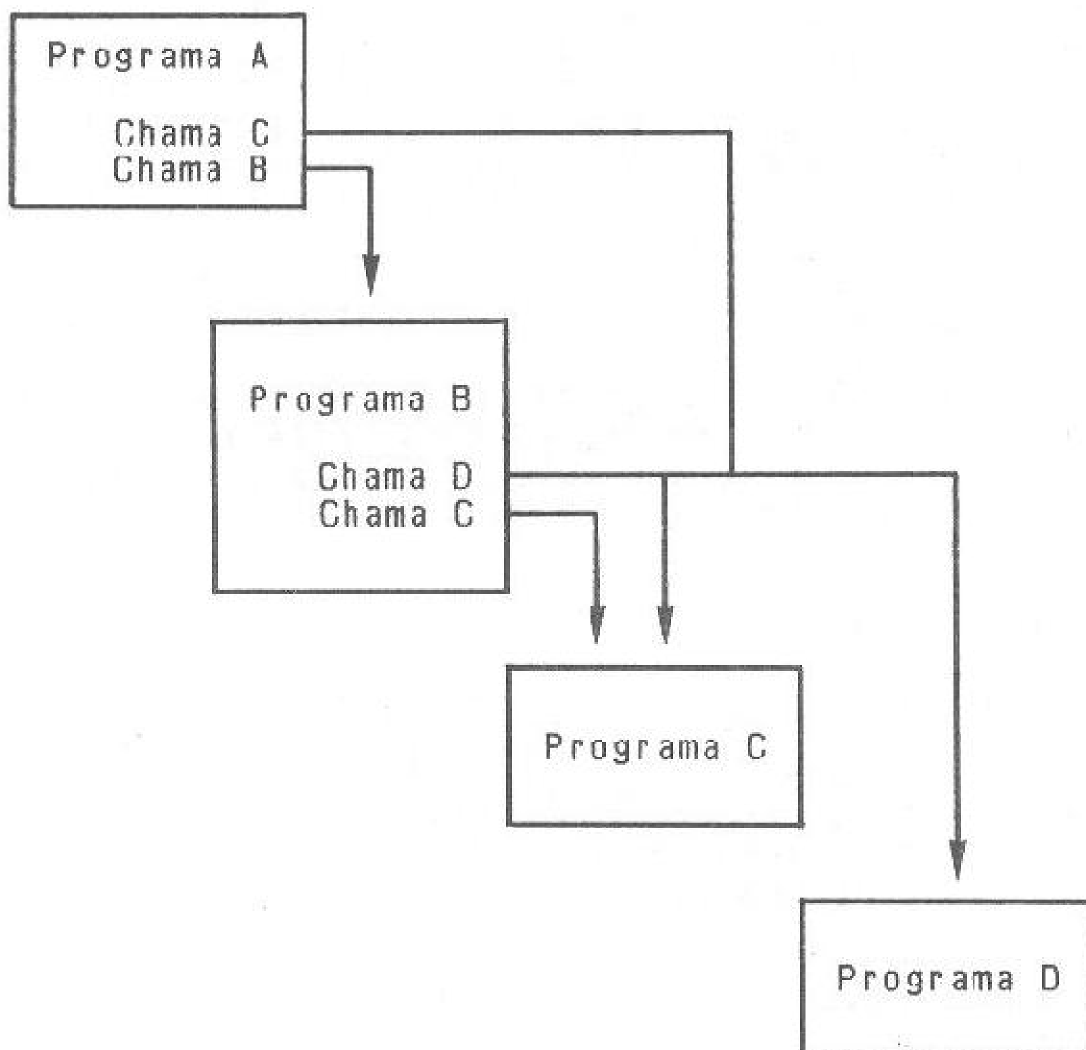
A divisão de um grande programa em vários outros menores deve ser feita segundos critérios específicos, de acordo com a necessidade e para isso dispõe-se de várias técnicas que a seguir relacionamos:

1 - Conexão Direta.

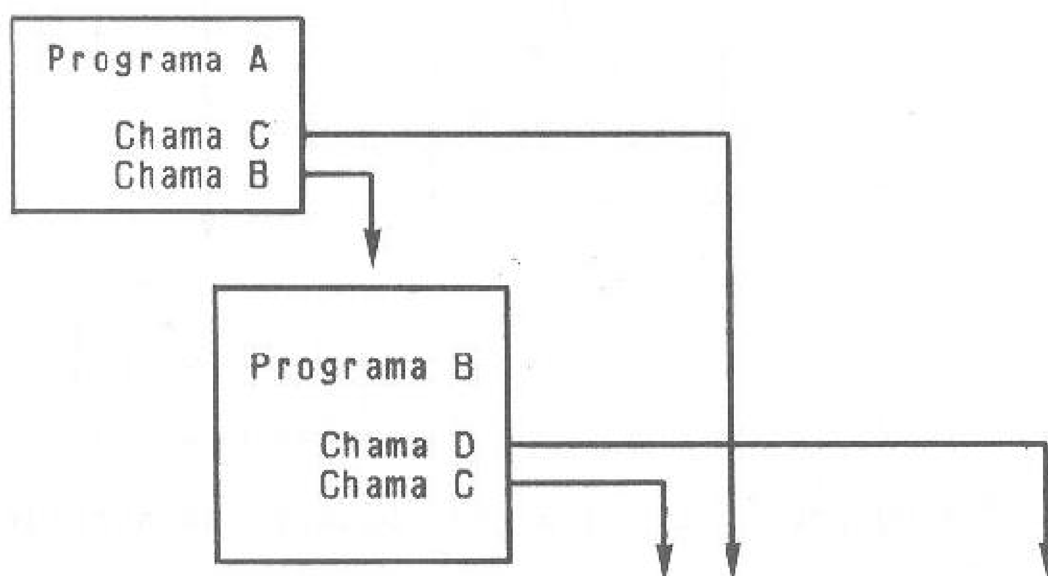
Nesta técnica, os programas ficam conectados diretamente, em função das necessidades do sistema. Assim, se um dos programas apresenta 2 alternativas e cada uma dessas alternativas é executada por um programa distinto, haverá uma chamada específica para cada um desses 2 programas:

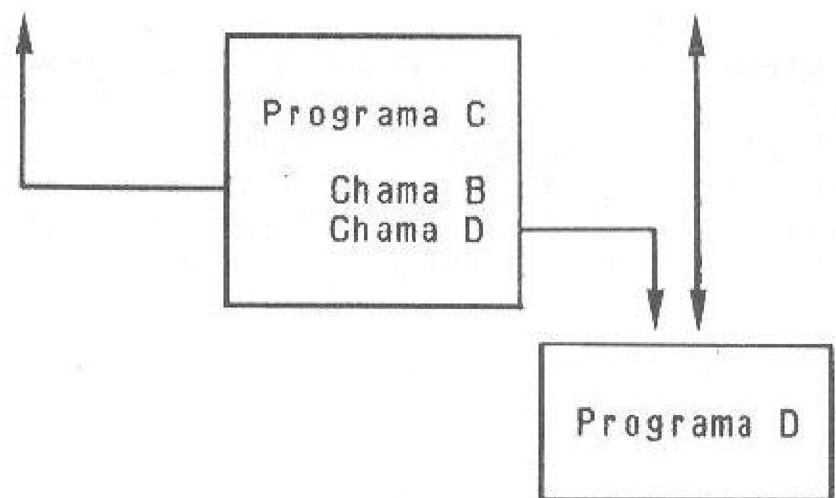


O programa B, por sua vez, poderá apresentar 2 outras opções:



O programa C, por sua vez, poderá apresentar outras 2 opções:





E, finalmente, o **programa** D poderá também apresentar alternativas:

Programa A

Chama C
Chama B

Programa B

Chama D
Chama C

Programa C

Chama B
Chama D

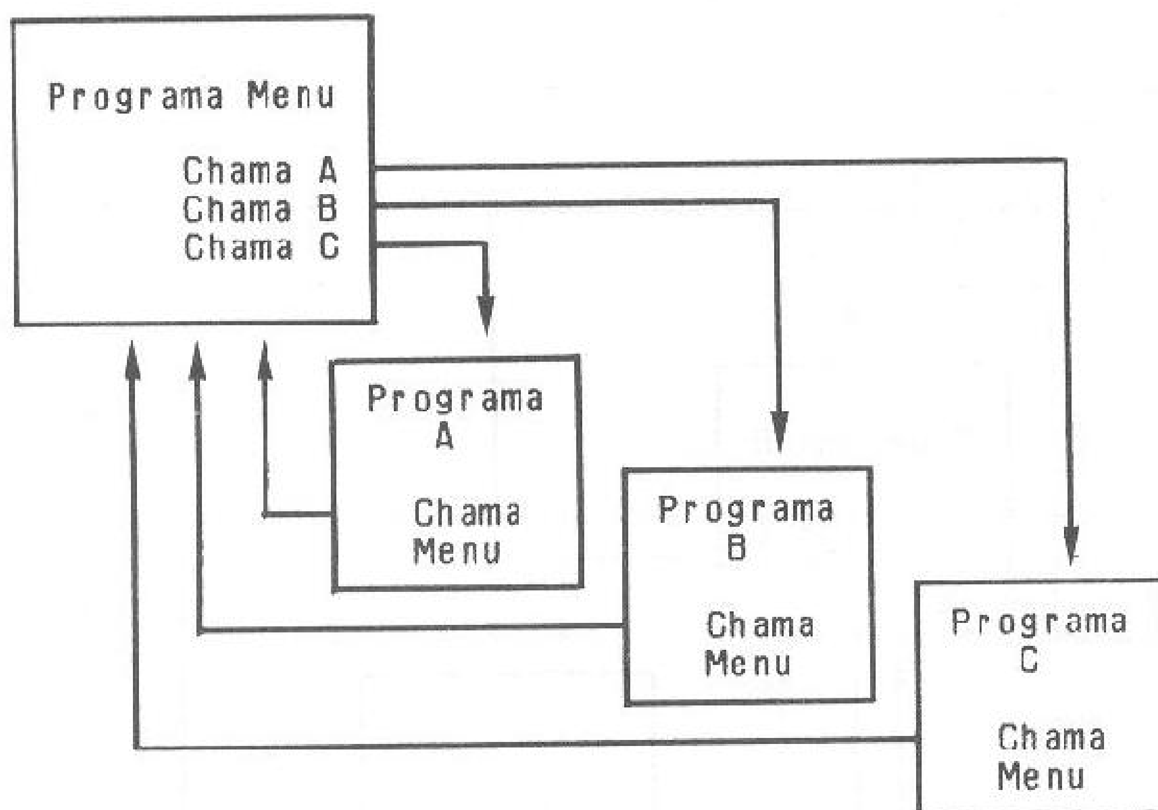
Programa D

Chama B
Chama A

A Conexão Direta apresenta a vantagem de o sistema acessar diretamente o programa desejado, agilizando a carga do programa, porém, na interpretação das conexões apresenta-se um tanto confusa, principalmente em grandes sistemas onde a presença de um número relativamente grande de programas poderá acarretar uma interpretação errada do sistema.

2 - Conexão em Estrela:

Neste esquema de conexão, existe um programa centralizador, que faz as vezes de programa de Cardápio de Programas:



A interpretação de um diagrama como este é bem mais simples, pois todas as chamadas a quaisquer dos programas é feita sempre pelo programa Menu e o retorno de qualquer dos programas é sempre feita para o programa Menu. Apresenta, porém, o incoveniente de, caso exista uma situação em que o programa B, por exemplo, necessite do programa C, a transferência deverá ser efetuada em duas etapas, isto é, inicialmente o programa B deverá chamar o programa Menu e em seguida o programa Menu deverá chamar o programa C.

Para permitir que um programa chame um outro, o BASIC oferece os seguintes comandos:

- RUN
- LOAD

Vejamos as formas de chamada em cada um desses casos:

1 - Comando RUN

O comando RUN é a alternativa normal empregada quando se deseja transferir o comando do sistema de um programa para outro.

Em geral, a escolha de qual dos programas é feita por um Menu de Opções, onde o usuário irá indicar qual o seu desejo. Em seguida, o programa escolherá a opção usando, por exemplo, o comando IF:

```
411 IF OPCAO=1 THEN RUN "A:PROGA"
412 IF OPCAO=2 THEN RUN "A:PROGB"
413 IF OPCAO=3 THEN RUN "A:PROGC"
```

Neste caso, ao executar o comando RUN, o programa que está na memória é apagado e no seu lugar é colocado o programa carregado.

Todas as variáveis do primeiro programa são perdidas, assim como, todos os arquivos abertos são sumariamente fechados.

2 - Comando LOAD.

O comando LOAD funciona de forma semelhante ao comando RUN, apresentando apenas uma diferença: Se o comando for executado com o parâmetro ",R" os arquivos porventura abertos no primeiro programa continuarão abertos no segundo, de modo que não existe a necessidade de se abrir novamente esses arquivos no início do programa chamado. O emprego desse comando é idêntico ao do comando RUN, isto é:

```
411 IF OPCAO=1 THEN LOAD "A:PROGA",R
412 IF OPCAO=2 THEN LOAD "A:PROGB",R
413 IF OPCAO=3 THEN LOAD "A:PROGC",R
```

Como em qualquer das alternativas acima as variáveis são perdidas, desejando-se passar os valores das variáveis de um programa para outro, deve-se criar um arquivo temporário e nele gravar o conteúdo das variáveis que necessitam ser passadas para o outro programa.

ESTRUTURA DOS ARQUIVOS

Um arquivo, como vimos no capítulo 4, pode ser do tipo sequencial, randômico ou um arquivo de programa.

Independentemente do seu tipo, o arquivo não fica disposto de forma contínua no disco.

O acesso ao disco é feito por módulos conhecidos como "setores".

No sistema MSX, cada setor tem normalmente capacidade para 512 bytes.

Deste modo, um arquivo com 2.000 bytes irá ocupar 4 setores do disco conforme o esquema seguinte:

512	Setor 1 (512 bytes)
512	Setor 2 (512 bytes)
512	Setor 3 (512 bytes)
464	Setor 4 (512 bytes)

Esses 4 setores não ficam necessariamente um ao lado do outro como seria de se imaginar. De acordo com a disponibilidade de espaço no disco, o sistema operacional irá alocar esses setores onde for mais conveniente, de modo que eles poderão estar, um no começo do disco, outro no meio e os demais no fim do disco.

Para compreendermos melhor o esquema de alocação de setores, vamos representar, de modo simplificado, a vida de um disquete.

1 - O disquete foi recém formatado, de modo que todos os setores estão livres.

2 - Nesse disquete foi definido um arquivo de 2.000 bytes.

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.000 bytes

3 - Vamos supor que nesse disco seja definido um segundo arquivo contendo 600 bytes.

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.000 bytes
Setor 5 Setor 6	Disponível 1.024 bytes	Arquivo 2 usa 600 bytes

4 - Agora, imagine que no Arquivo 1 sejam adicionados mais alguns registros e que após isso o tamanho do arquivo passe para 2.100 bytes.

Anteriormente o arquivo utilizava apenas 2.000 bytes embora tivesse reservado 2.048 bytes. Tinha, portanto, 48 bytes sobrando, sem uso.

Dos 100 novos bytes necessários 48 serão gravados nessa sobra que havia. E os 52 restantes, onde serão gravados ?

O sistema operacional reserva um novo setor de 512 bytes e grava nele esses 52 bytes restantes do arquivo. Mas onde está esse novo setor ?

O sistema procura o primeiro setor livre do disco, que no caso presente é o setor 7:

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.000 bytes
Setor 5 Setor 6	Disponível 1.024 bytes	Arquivo 2 usa 600 bytes
Setor 7	Disponível 512 bytes	Arquivo 1 (continuação) usa 52 bytes

5 - Caso, nesse disco seja gravado um Arquivo 3, com 800 bytes, o disco terá a seguinte ocupação:

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.048 bytes
Setor 5 Setor 6	Disponível 1.024 bytes	Arquivo 2 usa 600 bytes
Setor 7	Disponível 512 bytes	Arquivo 1 (continuação) usa 52 bytes
Setor 8 Setor 9	Disponível 1.024 bytes	Arquivo 3 usa 800 bytes

O fato de um arquivo ficar armazenado em setores não contíguos como o Arquivo 1, não traz nenhum incoveniente pois o sistema operacional sabe disso.

Quando um programa vai ler sequencialmente o Arquivo 1, o sistema operacional começa a leitura no primeiro setor que é o setor 1.

Terminada a leitura dos 512 bytes desse setor o próximo será lido. Quando o último byte do setor 4 for lido o sistema operacional saberá que deve ler o setor 7. Para isso ele guarda uma tabela onde estão relacionados os números dos setores de cada arquivo.

6 - Agora, imagine que o Arquivo 2 foi eliminado:

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.048 bytes
Setor 5 Setor 6	Livre 1.024 bytes	
Setor 7	Disponível 512 bytes	Arquivo 1 (continuação) usa 52 bytes
Setor 8 Setor 9	Disponível 1.024 bytes	Arquivo 3 usa 800 bytes

7 - Agora, imagine que novos registros foram adicionados ao Arquivo 1 e que ele passou a ter o tamanho de 2.800 bytes, de modo que os 5 setores anteriormente reservados para ele não sejam mais suficientes.

É claro que o sistema operacional irá reservar um novo setor para esse arquivo. Só que agora, em vez de reservar o setor 10, irá reservar o setor 5. O próximo setor a ser reservado é sempre o primeiro que está livre no disco.

Setor 1 Setor 2 Setor 3 Setor 4	Disponível 2.048 bytes	Arquivo 1 usa 2.048 bytes
Setor 5	Disponível 512 bytes	Arquivo 2 (continuação 2) usa 240 bytes.
Setor 6	Livre	
Setor 7	Disponível 512 bytes	Arquivo 1 (continuaçao 1) usa 512 bytes
Setor 8 Setor 9	Disponível 1.024 bytes	Arquivo 3 usa 400 bytes

O acesso ao Arquivo 1 será feito na seguinte ordem de setores:

1º setor 1
2º setor 2
3º setor 3
4º setor 4
5º setor 8
6º setor 5

Embora para o sistema operacional não faça nenhuma diferença a ordem dos setores, a eficiência do programa pode ser prejudicada se eles não forem contíguos.

Como vimos no capítulo 4, um disco normalmené composto de 40 trilhas por face e cada trilha comporta 9 setores. O disco tem, portanto, um total de 360 setores (5 1/4"!).

Considerando que cada setor ocupa 512 bytes e que o disco tem 2 faces, resulta que ele comporta 360 Kbytes ou seja 2x360 x 512 = 368.640 bytes.

Quando é feito um acesso ao disco, o tempo de acesso é composto por:

- Tempo de deslocamento da cabeça para a trilha desejada.

- Tempo de espera até a chegada do setor desejado. Enquanto o disco gira, os 9 setores da trilha ficam passando continuamente.

- Tempo de leitura ou de gravação quando todos os 512 bytes do setor são lidos ou gravados.

Desses tempos, o maior é o tempo de deslocamento da cabeça entre uma trilha e outra.

Quando um certo arquivo é antigo e durante a vida dele muitos registros foram sendo sucessivamente adicionados e ainda, outros arquivos foram sendo criados e eliminados no mesmo disco, pode resultar uma sequência de acesso do tipo:

1º setor	1
2º setor	2
3º setor	3
4º setor	171
5º setor	12
6º setor	246
7º setor	5

A leitura de todos os registros desse arquivo será relativamente lenta pois no meio das leituras a cabeça deverá ser deslocada muitas vezes de uma trilha para outra.

Recomenda-se, nestes casos, transferir, com o comando COPY, o arquivo para um novo disco, recém formatado. Nessa transferência o sistema operacional irá reservar setores consecutivos no novo disco, de modo que o arquivo que no disco antigo estava espalhado por setores esparsos, no novo disco ficará todo junto, em setores contíguos.

ESTRUTURA INTERNA DOS PROGRAMAS

A localização das instruções, dentro do programa pode afetar sensivelmente a velocidade de execução do programa.

O programa, embora seja uma sequência lógica onde existe um início e um fim, não necessita ter o início no princípio da listagem do programa. A primeira **instrução** de um programa pode perfeitamente ser uma instrução do tipo:

1 GOTO 7800

de modo que o início do programa estará no final da listagem.

Vejamos algumas recomendações para aumentar a velocidade de execução de um programa.

1 - Interpretador BASIC.

Durante a execução de um programa em BASIC, o interpretador analisa uma instrução de cada vez e somente após a execução completa da instrução é que ele passará para a interpretação da próxima instrução.

Como se sabe, as instruções, embora nas listagens sejam apresentadas uma abaixo das outras, na memória são dispostas em sequência, havendo o código **00 separando uma instrução da outra.**

Quando o interpretador encontra uma instrução do tipo:

GOTO 6274

ele não sabe, a priori, onde está a instrução 6274. Para encontrá-la, ele é obrigado a voltar para o início do programa e executar uma busca pela área de memória onde está o programa.

Isso consome tempo e, para os defensores ardorosos da programação estruturada, temos um argumento de ordem prática para evitar o uso abusivo do comando Goto.

2 - COMANDO DATA

O comando DATA define, em geral, uma grande quantidade de informacões, ocupando um longo trecho da memória.

Para o interpretador, é indiferente onde está localizado o comando DATA.

Se considerarmos o que foi visto no item anterior, veremos que a existência de um extenso comando DATA na parte inicial do programa fará com que as procuras pelos números das instruções acabem consumindo um tempo muito grande.

Recomenda-se, portanto, colocar todos os DATA's na parte final do programa.

3 - LOOP

Os ciclos, dentro de um programa são mais fáceis de serem visualizados quando se usam os comandos FOR...NEXT.

Lembre-se porém da possibilidade de se usar outros comandos,(IF, por exemplo) que, dependendo da condição do ciclo, pode ocasionar uma diminuição do tempo de execução.

4 - VARIÁVEL INTEIRA

Sempre que possível, empregue variáveis inteiras pois são mais rápidas de serem processadas.

Compare os tempos gastos pelos programas seguintes:

```
100 T = TIME
101 FOR N = 1 TO 1000
102 NEXT N
103 PRINT TIME - T
```

```
100 T = TIME
101 FOR N% = 1 TO 1000
102 NEXT N%
103 PRINT TIME - T
```

```
100 T =TIME
101 FOR N% = 1 TO 1000
102   A = 1
103 NEXT N%
104 PRINT TIME - T
```

```
100 T = TIME
101 FOR N% = 1 TO 1000
102 A% = 1
103 NEXT N%
104 PRINT TIME - T
```

5 - FIM DO LOOP

Existem situações dentro do ciclo FOR...NEXT, em que desejamos encerrar a execução do ciclo sem ter ainda atingido a condição final. Nestes casos, podemos igualar a variável de controle com a condição final.

```
100 FOR N = 1 TO 1000
101 PRINT N
102 IF N > 12 THEN N = 1000
103 NEXT N
104 PRINT "FIM"
```

6 - DEFINIÇÕES DE VARIÁVEIS

As variáveis do programa são definidas, isto é, um espaço para elas é alocado dinamicamente, na medida em que a execução do programa avança e é encontrada uma nova variável.

Se o programador prestar atenção ao mapa da memória apresentado no início deste capítulo, vai notar que as variáveis tipo Array são definidas acima das variáveis numéricas.

Executando o programa seguinte:

```
100 DIM E(100)
101 PRINT VARPTR(E(1))
102 B = 0
103 PRINT VARPTR(E(1))
104 C = 12
105 PRINT VARPTR(E(1))
```

O programador poderá observar que a cada nova variável que vai sendo definida no programa, o endereço onde estão as informações da matriz E(n) vai variando, o que significa que a cada nova definição, toda a matriz está sendo deslocada na memória, o que consome, evidentemente, algum tempo.

Para comprovar, execute os seguintes programas:

```
100 T = TIME
101 DIM E(1000)
102 B = 0
103 C = 10
104 PRINT TIME -T
```

```
100 T = TIME
101 B = 0
102 C = 10
103 DIM E(1000)
104 PRINT TIME - T
```

Recomenda-se definir todas as variáveis numéricas do programa antes da definição de qualquer matriz, mesmo que as variáveis sejam definidas com ZEROS.

COMPILAÇÃO DE PROGRAMAS

O BASIC Interpretado é aquele em que a execução do programa é feita após a interpretação de cada instrução. Isto é, em cada instrução é feita uma interpretação seguinda de uma execução.

O que é Interpretação ?

Interpretar significa traduzir a instrução que está montada segundo a sintaxe da linguagem BASIC para uma instrução em linguagem do Z80, para ser executada.

O microprocessador só entende programas do tipo:

```
11 03 00    LD   DE,0003
CD 50 AB    CALL CONOUT
17          RLA
30 FA       JR   NC,VOLTA
1F          RRA
C9          RET
```

que são programas em linguagem de máquina.

O interpretador converte instruções em BASIC para instruções em Linguagem de Máquina e executa-as logo a seguir. Após a execução, pega a próxima instrução BASIC e repete o ciclo.

Usando-se o BASIC Interpretado, na execução do programa estará embutido o tempo de conversão.

O Compilador é um utilitário que converte não apenas uma instrução mas o programa todo de BASIC para Linguagem de Máquina sem executá-lo.

A entrada para o Compilador é o programa em BASIC e a saída é o programa já convertido em um programa em Linguagem de Máquina.

O programa poderá ser executado após a compilação. Neste caso, não haverá mais o tempo de interpretação pois todas as instruções já estarão devidamente interpretadas, de modo que a execução desse programa compilado será muito mais rápida.

CRIATTIVIDADE

Embora tenha sido apresentada ao longo deste livro uma série de dicas para agilizar a execução de um programa, o que vale mesmo é a criatividade do programador em criar **algoritmos** inteligentes capazes de resolver o problema com eficiência, mesmo que a sua aparência não dê essa impressão.

Experimente executar o programa abaixo e analise o que faz o algoritmo:

```
100 DEFINT I-M
102 PRINT "Fornecer um numero impar maior que 3."
104 INPUT "Tamanho";M
106 GOSUB 200
108 PRINT "Experimente somar as Horizontais,
          Verticais e Diagonais:"
110 FOR I = 1 TO M
112   FOR J = 1 TO M
114     PRINT TAB(3*J)USING"###";A(I,J);
116     B = B + A(I,J)
118   NEXT J
120   PRINT "=";B : B = 0
122 NEXT I
124 END
200 '---------
202 'ALGORITMO
204 '---------
206 DIM A(M,M)
208 K = M / 2
210 L = 1
212 I = I + K
```

```
214 J = J + 1
216 IF L > M*M THEN RETURN
218 IF I > M THEN I = I - M
220 IF J > M THEN J = J - M
222 IF I < 1 THEN I = I + M
224 IF J < 1 THEN J = J + M
226 IF A(I,J) = 0 THEN A(I,J) = L :
                        L = L + 1 :
                        GOTO 212 :
                        ELSE I = I + K:
                        J = J - 1 :
                        GOTO 218
```

Agora esqueça o **algoritmo** apresentado e procure desenvolver um **algoritmo** que resolva o seguinte problema:

"Preencher uma matriz 5 X 5 com números sequenciais, começando pelo 1, sem repetir, de forma que a soma dos números dispostos em qualquer das horizontais, verticais ou diagonais, resulte sempre em 65".

PROTEÇÃO DE PROGRAMAS

O programador que pretende profissionalizar-se deve, além das preocupações apresentadas neste livro, preocupar-se com o esquema de distribuição dos programas.

Não é raro encontrar programadores lamentando-se do fato de uma pessoa inescrupulosa estar ganhando muito dinheiro comercializando cópias de programas sem a autorização dos autores.

Embora haja uma legislação sobre a matéria, nem sempre a mesma é obedecida e é quase impossível descobrir em que empresas estão as cópias ilegais de nossos programas.

Muitos programadores, na tentativa de dificultar ou mesmo impedir a cópia ilegal dos programas, desenvolvem técnicas especiais de fazer uma gravação magnética ou física nos disquetes. Essas proteções, no entanto, tem uma duração efemêra pois após algum tempo, algum programador pirata irá descobrir como é feita a proteção e criará um Eliminador de Proteção capaz de produzir cópias facilmente copiáveis.

Não perca muito tempo tentando desenvolver alguma técnica super-elaborada para a proteção do seu

programa pois no mercado encontramos dezenas de programas que apresentam uma baixa eficiência justamente por causa da técnica de proteção e mesmo assim são amplamente pirateados.

O melhor critério é relacionar-se comercialmente somente com pessoas ou empresas idôneas e em cada venda providenciar um Contrato de Licença de Uso do Programa.

Nos países onde há uma grande produção de programas houve época em que se tentou proteger os programas por meio de técnicas de proteção gravadas nos disquetes, mas a tendência, hoje em dia, é de preocupar-se com a idoneidade dos compradores.

9

exemplo de um sistema

Este capítulo contém um exemplo prático de um sistema completo de programas para controle do estoque de materiais do almoxarifado de uma pequena empresa.

O exemplo controla apenas a parte física, isto é, as quantidades de materiais, não entrando no mérito dos preços dos mesmos.

A título de exercício, o programador poderá extender o controle para controlar os pagamentos (Contas a Pagar). Para isso deverá montar um arquivo para o Cadastro de Fornecedores e outro para o Registro e Acompanhamento de pagamento das duplicatas.

Poderá também extender o controle para controlar o Custo de Consumo dos diversos setores da empresa. Para isso deverá montar um arquivo para o Cadastro de Setores e outro para o Registro de Saída para os setores.

Este sistema é a continuação da análise realizada no capítulo 7, onde são definidos os seguintes programas:

1 - AUTOEXEC.BAS
2 - COMAT.BAS
3 - CMCAD.BAS
4 - CMMOV.BAS
5 - CMREGENT.BAS
6 - CMREGSAI.BAS
7 - CMALTENT.BAS
8 - CMALTSAI.BAS
9 - CMCANENT.BAS
10 - CMCANSAI.BAS
11 - CMREL.BAS
12 - CMREL1.BAS
13 - CMREL2.BAS
14 - CMREL3.BAS
15 - CMREL4.BAS
16 - CMREL5.BAS
17 - CMENC.BAS
18 - CMMAN.BAS

ESPECIFICAÇÃO DOS PROGRAMAS

A seguir, comentamos brevemente o que faz e a que se destina cada programa de nosso sistema.

AUTOEXEC.BAS

Realiza a apresentação do sistema, quando solicita que seja digitada a data de acesso.
Carrega e executa o programa COMAT.BAS .

COMAT.BAS

Apresenta as opções do sistema, podendo carregar e executar um dos seguintes programas:

- CMCAD.BAS para cadastrar
- CMMOV.BAS para movimentar materiais
- CMREL.BAS para emitir os relatorios
- CMENC.BAS para encerrar o mes
- CMMAN.BAS para efetuar a manutenção do sistema

CMCAD.BAS

Permite efetuar a inclusão de novos materiais no cadastro. Após a inclusão, carrega de volta o programa COMAT.BAS .

CMMOV.BAS

Permite realizar a movimentação de materiais, carrega e executa um dos seguintes programas:

- CMREGENT.BAS para dar entrada
- CMREGSAI.BAS para dar saída
- CMALTENT.BAS para alterar uma entrada
- CMALTSAI.BAS para alterar uma saída
- CMCANENT.BAS para cancelar uma entrada
- CMCANSAI.BAS para cancelar uma saída

Após a execução, carrega de volta o programa COMAT.BAS .

CMREL.BAS

Permite escolher um dos relatórios programados, feito por um dos programas:

- CMREL1.BAS para emitir o relatório RELAÇÃO DE MATERIAIS .
- CMREL2.BAS para emitir o relatório SALDOS DO ESTOQUE .
- CMREL3.BAS para emitir o relatório MOVIMENTO DO ALMOXARIFADO .
- CMREL4.BAS para emitir o relatório SAÍDAS DO ALMOXARIFADO .
- CMREL5.BAS para emitir o relatório POSIÇÃO DO ESTOQUE .

Após a execução, carrega de volta o programa COMAT.BAS .

CMENC.BAS

Permite proceder ao encerramento do mes, quando os saldos são atualizados.

Após a execução, carrega de volta o programa COMAT.BAS .

CMMAN.BAS

Permite consultar e alterar qualquer um dos dados do arquivo de parâmetros do sistema. Após a execução. carrega de volta o programa COMAT.BAS .

CMREGENT.BAS

Permite efetuar o registro de uma entrada no estoque. Após a execução, carrega de volta o programa CMMOV.BAS .

CMREGSAI.BAS

Permite efetuar o registro de uma saída do estoque. Após a execução, carrega de volta o programa CMMOV.BAS .

CMALTENT.BAS

Permite alterar os dados de uma entrada registrada anteriormente. Após a execução, carrega de volta o programa CMMOV.BAS .

CMALTSAI.BAS

Permite alterar os dados de uma saída registrada anteriormente. Após a execução, carrega de volta o programa CMMOV.BAS .

CMCANENT.BAS

Permite cancelar uma entrada registrada anteriormente. Após a execução, carrega de volta o programa CMMOV.BAS .

CMCANSAI.BAS

Permite cancelar uma saída registrada anteriormente. Após a execução, carrega de volta o programa CMMOV.BAS .

CMREL1.BAS

Emite o relatório Relação de Materiais. Após a execução, carrega de volta o programa CMREL.BAS .

CMREL2.BAS

Emite o relatório Saldos do Estoque. Após a execução, carrega de volta o programa CMREL.BAS .

CMREL3.BAS

Emite o relatório Movimento do Almoxarifado. Após a execução carrega de volta o programa CMREL.BAS.

CMREL4.BAS

Emite o relatório Saídas do Almoxarifado. Após a execução carrega de volta o programa CMREL.BAS.

CMREL5.BAS

Emite o relatório Posição do Estoque. Após a execução, carrega de volta o programa CMREL.BAS .

ESPECIFICAÇÃO DOS ARQUIVOS

Cadastro de Materiais

- nome: CMCAD.ARQ
- tipo: Randômico

- Código do Material	CD$	4
- Descrição do Material	DC$	25
- Tipo do Material	TC$	10
- Saldo do Período Anterior	SR$	4
- Saldo Atual do Estoque	SL$	4
- Quantidade Mínima no Estoque	QM$	4
- Data da Última Entrada	UE$	6
- Data da Última Saída	US$	6

Total = 63 bytes

Arquivo de Movimento

- nome: CMMOV.ARQ
- tipo: Randômico

- Data do Movimento	MD$	6
- Histórico	MH$	10
- Código do Material Movimentado	MC$	4
- Quantidade Movimentada	MO$	4
- Tipo do Movimento (E/S)	MV$	1

Total = 25 bytes

Arquivo de Parâmetros do Sistema

- nome: CMPAR.ARQ
- tipo: Randômico

- Nome do Sistema	TT$	30
- Número do Mes em Andamento	NM$	2
- Nome do Mes em Andamento	DM$	9
- Quantidade de Registros no Cadastro	RC$	2
- Quantidade de Registros no Movimento	RM$	2
- Ultima data Acessada: Dia	UD$	2
Mes	UM$	2
Ano	UA$	2

Total = 51 bytes

MODELO DOS RELATóRIOS IMPRESSOS

Existe uma infinidade de relatórios possíveis para atender a cada situação específica de necessidade de informações do estoque.

Tratando-se de um exemplo prático onde procura-se consolidar os conhecimentos apresentados nos capítulos anteriores, os relatórios aqui descritos foram selecionados de forma a consolidar as várias técnicas de programação e de acesso aos arquivos.

Desse modo, os programas de emissão de relatórios apresentam uma complexidade crescente, na tentativa de demonstrar as técnicas para se obter:

1 - Simples Relações

São programas fáceis de serem elaborados pois operam em geral com um único arquivo.

Como exemplo de um programa desse tipo é apresentado o programa CMREL1.BAS que imprime o relatório Relação de Materiais.

```
             1         2         3         4
    1234567890123456789012345678901234567890123
   ------------------------------------------------
01 |Metalurgica METAL Ltda.
02 |
03 |Controle de Materiais - MATERIAIS      FL-XXX
04 |
05 |Data da Emissao: xx/xx/xx
06 |-------------------------------------------
07 |CODIGO DESCRICAO                      TIPO
08 |
09 | xxxx  xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx
10 |
```

2 - Relação com Totalização

São programas fáceis de serem elaborados pois operam sobre um arquivo, totalizando certas quantidades em um acumulador que é impresso no final do relatório.

Como exemplo de um programa desse tipo é apresentado o programa CMREL2.BAS que emite o relatório Saldos do Estoque.

```
             1         2         3         4         5
    123456789012345678901234567890123456789012345678901
   ____________________________________________________
01 |Metalurgica METAL Ltda.
02 |
03 |Controle de Materiais - SALDOS DO ESTOQUE    FL-XXX
04 |
05 |Data da Emissao: XX/XX/XX
06 |---------------------------------------------------
07 |CODIGO DESCRICAO                  SALDO ULT.ENTRADA
08 |
09 | XXXX  XXXXXXXXXXXXXXXXXXXXXXXXX XXXXX    XX/XX/XX
10 | XXXX  XXXXXXXXXXXXXXXXXXXXXXXXX XXXXX    XX/XX/XX
11 | XXXX  XXXXXXXXXXXXXXXXXXXXXXXXX XXXXX    XX/XX/XX
63 |---------------------------------------------------
64 |Quantidade de Itens no Cadastro= XXXX
65 |===================================================
```

3 - Relação Cruzada.

São programas ainda fáceis de serem elaborados pois operam basicamente sobre um arquivo, buscando em outro a informação complementar.

Como exemplo de um programa desse tipo é apresentado o programa CMREL3.BAS que emite o relatório Movimento do Almoxarifado.

```
             1         2         3         4         5
    12345678901234567890123456789012345678901234567890123456789012345678
   __________________________________________________________________________
01 |Metalurgica METAL Ltda.
02 |
03 |Controle de Materiais - MOVIMENTO DO ALMOXARIFADO FL-xxx
04 |
05 |Data da Emissao: xx/xx/xx
06 |--------------------------------------------------------------------------
07 | DATA    DESCRICAO                    DOCUMENTO ENTRADA SAIDA
08 |
09 |xx/xx/xx xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx xxxxx  xxxxx
10 |         xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx xxxxx  xxxxx
11 |         xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx xxxxx  xxxxx
12 |
13 |xx/xx/xx xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx xxxxx  xxxxx
14 |         xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxx xxxxx  xxxxx
15 |
16 |
65 |==========================================================================
```

4 - Relação Ordenada-I.

São programas que imprimem relatórios ordenados, sem, no entanto, ordenar o arquivo.

Como exemplo de um programa desse tipo é apresentado o programa CMREL4.BAS que emite o relatório Saídas do Almoxarifado.

```
             1         2         3         4
    123456789012345678901234567890123456789012345
   +---------------------------------------------
01 |Metalurgica METAL Ltda.
02 |
03 |Controle de Materiais - SAIDAS         FL-xxx
04 |
05 |Data da Emissao: xx/xx/xx
06 |---------------------------------------------
07 | DATA     DESCRICAO                QUANTIDADE
08 |
09 |xx/xx/xx xxxxxxxxxxxxxxxxxxxxxxxxx    xxxxx
10 |xx/xx/xx xxxxxxxxxxxxxxxxxxxxxxxxx    xxxxx
11 |xx/xx/xx xxxxxxxxxxxxxxxxxxxxxxxxx    xxxxx
12 |
65 |=============================================
```

5 - Relação Ordenada-II.

São programas que efetivamente ordenam o arquivo. Como exemplo de um programa desse tipo é apresentado o programa CMREL5.BAS que emite o relatório Posição do Estoque.

```
             1         2         3         4         5         6
    123456789012345678901234567890123456789012345678901234567890123456
   +------------------------------------------------------------------
01 |Metalurgica METAL Ltda.
02 |
03 |Controle de Materiais - POSICAO DO ESTOQUE FL-xxx
04 |
05 |Data da Emissao: xx/xx/xx
06 |------------------------------------------------------------------
07 |DESCRICAO                          ANTERIOR ENTRADAS SAIDAS ATUAL
08 |
09 |xxxxxxxxxxxxxxxxxxxxxxxxx            xxxxx    xxxxx   xxxxx xxxxx
10 |xxxxxxxxxxxxxxxxxxxxxxxxx            xxxxx    xxxxx   xxxxx xxxxx
11 |xxxxxxxxxxxxxxxxxxxxxxxxx            xxxxx    xxxxx   xxxxx xxxxx
65 |==================================================================
```

DESENHO DAS TELAS DE TRABALHO

A seguir estão relacionados os desenhos de todas as telas de trabalho do sistema de controle de materiais.

PROGRAMA: AUTOEXEC.BAS

```
  0123456789012345678901234567890123456789
 0COMAT - Controle de Materiais
 1------> ABERTURA <----------------------
 2
 3
 4Seja benvindo ao sistema de Controle de
 5Materiais
 6
 7Favor fornecer a data de hoje:dd/mm/aa
 8
 9
10
11
12
13
14
```

TODOS OS PROGRAMAS

```
  0123456789012345678901234567890123456789
 0COMAT - Controle de Materiais
 1------> TRANSFERENCIA <-----------------
 2
 3
 4Estou providenciando a carga do
 5programa COMAT.BAS
 6
 7Favor aguardar alguns instantes.
 8
 9
10
11
12
13
14
15
16
```

PROGRAMA: COMAT.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> CARDAPIO DE OPÇOES <----------


Escolher uma das opcoes:

1 - Cadastrar um novo material
2 - Movimentar
3 - Emitir um relatorio
4 - Encerrar o mes
5 - Manutencao do Sistema
6 - Encerrar o Processamento

*** Opcao escolhida = █










color  auto   goto    list    run
```

PROGRAMA CMCAD.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
-----> CADASTRAMENTO MATERIAIS <-------


Codigo:0007

Descricao:PAPEL LINHA D'AGUA 120g
Tipo SIMETRIK
Saldo: 200.3
Minimo: 100

Posso Gravar no Disco? (S/N) S
         **********************
         Estou providenciando a
         insercao do registro.
            NAO INTERROMPA !
         **********************
     0%             50%             100%
     !    !    !    !    !    o     !
         >    >    >    >    >



color 1 cload"  cont    list    run
```

PROGRAMA: CMMOV.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
-------> MOVIMENTO DE MATERIAIS <-------


Escolher uma das opcoes:

1 - Registrar uma Entrada
2 - Registrar uma Saida
3 - Alterar os Dados de uma Entrada
4 - Alterar os Dados de uma Saida
5 - Cancelar uma Entrada
6 - Encerrar o Processamento
7 - Voltar a Tela Anterior

*** Opcao escolhida = 0









color    auto    goto    list    run
```

PROGRAMA: CMREGENT.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> REGISTRO DE ENTRADAS <----------


Data da Entrada █ /  /
Codigo do Material:


Descricao:
Saldo no Estoque:

Quantidade da Entrada:
Historico:

Posso Gravar no Disco? (S/N)








color    auto    goto    list    run
```

PROGRAMA: CMREGSAI.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> REGISTRO DE SAIDAS <-----------


Data da Saida   :█/  /  :
Codigo do Material:      :


Descricao:
Saldo no Estoque:        :

Quantidade da Saida   :    :
Historico:               :

Posso Gravar no Disco? (S/N): :








color    auto     goto     list     run
```

PROGRAMA: CMALTENT.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> ALTERACAO DE ENTRADA <----------


Data da Entrada :█/  /  :
Codigo do Material:      :


Descricao:
Saldo no Estoque:        :

Quantidade da Entrada:     :
Historico:               :

Posso Gravar no Disco? (S/N): :








color    auto     goto     list     run
```

PROGRAMA: CMALTSAI.BAS

```
  0123456789012345678901234567890123456789
 0 COMAT - Controle de Materiais
 1 ------> ALTERACAO DE SAIDA <-----------
 2
 3
 4 Data da Saida   :█ /  /  :
 5 Codigo do Material:      :
 6
 7
 8 Descricao:
 9 Saldo no Estoque:        :
10
11 Quantidade da Saida  :      :
12 Historico:
13
14 Posso Gravar no Disco? (S/N): :
15
16
17
18
19
20
21
22
23 color   auto    goto    list    run
```

PROGRAMA: CMCANENT.BAS

```
  0123456789012345678901234567890123456789
 0 COMAT - Controle de Materiais
 1 ------> CANCELAMENTO DE ENTRADA <-------
 2
 3
 4 Data de entrada:█ /  /  :
 5 Codigo do material:     :
 6
 7
 8 Descricao:                          :
 9 Saldo no Estoque:       :
10
11 Quantidade na entrada:       :
12 Historico:                          :
13
14
15
16
17
18
19
20
21
22
23
```

PROGRAMA: CMCANSAI.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> CANCELAMENTO DE SAIDA <---------


Data de saida      :  /  /  :
Codigo no estoque  :        :


Descricao:                              :
Saldo no estoque:         :

Quantidade na saida      :        :
Numero da requisicao:             :
```

PROGRAMA: CMREL.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> EMISSAO DE RELATORIOS <---------


Escolha uma das opcoes seguintes:

1 - Relacao de Materiais
2 - Saldos do Estoque
3 - Movimento do Almoxarifado
4 - Saidas do Almoxarifado
5 - Posicao do Estoque
6 - Voltar a tela anterior


Opcao escolhida: :
```

PROGRAMA: CMENC.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> ENCERRAMENTO DO MES <---------

Mes em andamento :Abril      :

Confirmar o encerramento (S/N):

Os saldos atuais foram transferidos para

os saldos anteriores.

Os arquivos estao prontos para receberem

os dados doo mes de Janeiro






Pressione qquer tecla para continuar ...



color    auto     goto     list     run
```

PROGRAMA CMMAN.BAS

```
0123456789012345678901234567890123456789
COMAT - Controle de Materiais
------> ARQUIVO DE PARAMETROS <--------


Nome do sistema:
     COMAT - Controle de Materiais

Número do mês :  :
Descrição do mês :Janeiro   :
Quantidade de Itens no Cadastro : 16
Quantidade Movimentada : 5    :

Ultima data acessada 10/10/77:

Alterar no disco? (S/N): :








color    auto     goto     list     run
```

PROGRAMA: COMAT.BAS

```
  0123456789012345678901234567890123456789
 0 COMAT - Controle de Materiais
 1 ------> ENCERRAMENTO <-----------------
 2
 3
 4 O Sistema de Controle de Materiais
 5 esta sendo desativado.
 6
 7 Ate a proxima !
 8 Ok
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

Para desenhar suas próprias telas use um mapa como o mostrado a seguir!

```
  0123456789012345678901234567890123456789
 0 ........................................
 1 ........................................
 2 ........................................
 3 ........................................
 4 ........................................
 5 ........................................
 6 ........................................
 7 ........................................
 8 ........................................
 9 ........................................
10 ........................................
11 ........................................
12 ........................................
13 ........................................
14 ........................................
15 ........................................
16 ........................................
17 ........................................
18 ........................................
19 ........................................
20 ........................................
21 ........................................
22 ........................................
23 ........................................
```

LISTAGEM DOS PROGRAMAS

Afim de diminuir o espaço ocupado pelos programas, os trechos desenvolvidos nos capítulos anteriores não estão listados.

Tais trechos são os seguintes:

NOME	DESENVOLVIDO NO:	
ESPERE.FNT	capítulo 2	página 25
QQUER.FNT	capítulo 3	página 50
SONS.FNT	capítulo 3	página 51
ROTENT.FNT	capítulo 3	página 52
ABREARQ.FNT	capítulo 4	página 79
ROTINS.FNTT	capítulo 4	página 86
ROTPMOV.FNT	capítulo 4	página 105
ROTPROC.FNT	capítulo 4	página 107
MAPAACC.FNT	capítulo 4	página 113
ERRO.FNT	capítulo 5	página 122

Dos programas do sistema, apenas os indicados estão listados, de modo que os demais deverão ser desenvolvidos pelo programador a título de exercício.

PROGRAMA	LISTAGEM	
1 - AUTOEXEC.BAS	SIM	
2 - COMAT.BAS	SIM	
3 - CMCAD.BAS	SIM	
4 - CMMOV.BAS	SIM	
5 - CMREGENT.BAS	SIM	
6 - CMREGSAI.BAS		NAO
7 - CMALTENT.BAS	SIM	
8 - CMALTSAI.BAS		NAO
9 - CMCANENT.BAS		NAO
10 - CMCANSAI.BAS		NAO
11 - CMREL.BAS		NAO
12 - CMREL1.BAS	SIM	
13 - CMREL2.BAS	SIM	
14 - CMREL3.BAS	SIM	
15 - CMREL4.BAS	SIM	
16 - CMREL5.BAS	SIM	
17 - CMENC.BAS	SIM	
18 - CMMAN.BAS	SIM	

Programa AUTOEXEC.BAS

```
100 ' ---------------------------------------------
102 ' AUTOEXEC.BAS = Inicia Controle de Materiais
104 ' ---------------------------------------------
106 '
108 ' R.M.Watanabe                    23/04/87
200 ' -------------------
202 ' Parametros Iniciais
204 ' -------------------
206 DR$ = "A:"
208 NARQ = 1
210 GOSUB 6010:GET #1,1
220 DIM HC(3),VC(3),ZC(3)
222 HC(1)=30:VC(1)=7:ZC(1)=2
224 HC(2)=33:VC(2)=7:ZC(2)=2
226 HC(3)=36:VC(3)=7:ZC(3)=2
228 CR = 1
400 ' ------------
402 ' Desenha Tela
404 ' ------------
410 CLS
412 PRINT TT$
414 PRINT "------> ABERTURA <----------------------"
416 PRINT
420 PRINT "Seja benvindo ao sistema de Controle de"
422 PRINT "Materiais"
424 PRINT
426 PRINT "Favor fornecer a data de hoje:dd/mm/aa"
600 ' -------------
602 ' Recebe a Data
604 ' -------------
610 LOCATE HC(CR),VC(CR)
612 CS=ZC(CR):GOSUB 1000
620 IF ASC(A$) = 13 THEN GOTO 710
622 IF ASC(A$) = 28 THEN GOTO 710
624 IF ASC(A$) = 29 THEN GOTO 720
626 IF ASC(A$) = 27 THEN GOTO 900
628 GOTO 730
710 ' ------> Avanca o Cursor
712           CR = CR + 1
714           IF CR > 3 THEN GOTO 850
716           GOTO 600
720 ' ------> Retrocede o Cursor
722           CR = CR - 1
724           IF CR < 1 THEN CR = 1
726           GOTO 600
```

```
730 ' ------> Desvia
732             ON CR GOSUB 810,820,830
734             CR = CR + 1
736             IF CR > 3 THEN GOTO 850
738             GOTO 600
800 ' ----------------------
802 ' Opcoes conforme o Cursor
804 ' ----------------------
810 '
812 DD$ = A$
814 RETURN
820 '
822 MM$ = A$
824 RETURN
830 '
832 AA$ = A$
834 RETURN
850 ' -----------
852 ' Consistencia
854 ' -----------
856   SEGUE$ = "SIM"
858      IF VAL(DD$) < 1 THEN CR=1:SEGUE$="NAO"
860      IF VAL(DD$) > 31 THEN CR=1:SEGUE$="NAO"
862      IF VAL(MM$) < 1 THEN CR=2:SEGUE$="NAO"
864      IF VAL(MM$) > 12 THEN CR=2:SEGUE$="NAO"
870      IF SEGUE$ <> "SIM" THEN GOSUB 5010:
         GOSUB 5300:GOTO 600
880 LSET UD$ = DD$
882  LSET UM$ = MM$
884   LSET UA$ = AA$
886    PUT #1,1
888     CLOSE #1
890      PP$ = "COMAT.BAS"
892       GOSUB 5100
900 ' ----------
902 ' Desistencia
904 ' ---------
906   PRINT
908   PRINT
910   END
```

```
MERGE "ROTENT.FNT"
MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "QQUER.FNT"
MERGE "ERRO.FNT"
MERGE "ABREARQ.FNT"
DELETE 6020-6036
SAVE "AUTOEXEC.BAS"
```

Programa COMAT.BAS

```
100 ' ===================================
102 ' COMAT.BAS = Controle de Materiais
104 ' ===================================
106 ' R.M.Watanabe                06/04/87
200 ' -------------------
202 ' Parametros Iniciais
204 ' -------------------
206     DR$ = "A:"
208     NARQ = 1
210     GOSUB 6010:GET #1,1
400 ' ------------------
402 ' Formatacao da tela
404 ' ------------------
406 CLS
408 PRINT TT$
410 PRINT"-------> CARDPIO DE OPöES <-----------"
412 PRINT
414 PRINT "Escolher uma das opcoes:"
416 PRINT
418 PRINT "1 - Cadastrar um novo material"
420 PRINT "2 - Movimentar"
422 PRINT "3 - Emitir um relatorio"
424 PRINT "4 - Encerrar o mes"
426 PRINT "5 - Manutencao do Sistema"
428 PRINT "6 - Encerrar o Processamento"
430 PRINT
432 PRINT "*** Opcao escolhida = ";CHR$(255);
500 ' ----------------
501 ' Entrada da Opcao
502 ' ----------------
510     K$=INKEY$:IF K$="" THEN GOTO 510
520      IF K$<"1" OR K$>"6" THEN GOTO 510
530       PRINT
540        PRINT "A opcao escolhida foi:"K$
550         IF K$="6" THEN GOTO 760
560          OPCAO = VAL(K$)
```

```
690 ON OPCAO GOTO 710,720,730,740,750,760
710 ' -----------------------
712 ' OPCAO = 1  Cadastramento
714 ' -----------------------
716   PP$="CMCAD.BAS":GOSUB 5100
720 ' ----------------------
722 ' OPCAO = 2  Movimentacao
724 ' ----------------------
726   PP$="CMMOV.BAS":GOSUB 5100
730 ' ---------------------
732 ' OPCAO = 3  Relatorios
734 ' ---------------------
736   PP$="CMREL.BAS":GOSUB 5100
740 ' -------------------------
742 ' OPCAO = 4  Encerrar o mes
744 ' -------------------------
746   PP$="CMENC.BAS":GOSUB 5100
750 ' ---------------------
752 ' OPCAO = 5  Manutencao
754 ' ---------------------
756   PP$="CMMAN.BAS":GOSUB 5100
760 ' -----------------------
762 ' OPCAO = 6  Encerramento
764 ' -----------------------
766 PRINT "########################################"
768 PRINT " O Sistema de Controle de Materiais esta
          sendo desativado ..."
770 PRINT
772 PRINT" Ate a proxima!"
774 PRINT
776 PRINT "########################################"
778 END

MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "ABREARQ.FNT"
DELETE 6020-6036
SAVE "COMAT.BAS"
```

Programa CMCAD.BAS

```
1 MAXFILES = 2
100 ' ========================
102 ' CMCAD.BAS = Cadastramento
104 ' ========================
106 ' Este programa efetua o cadastra
108 ' mento de novos materiais.
112 '
```

```
114 ' R.M.Watanabe 07/04/87
116 '
200 ' --------------------
201 ' Parametros Iniciais
202 ' --------------------
204 DR$ = "A:"
206 NARQ = 1
208 GOSUB 6010:GET #NARQ,1
210 MX% = CVI(RC$)
212 NARQ = 2:GOSUB 6020
216   EC$=SPACE$(4)
218   ED$=SPACE$(25)
220   ET$=SPACE$(10)
222   ES$=SPACE$(5)
224   EM$=SPACE$(5)
230 DIM HC(6),VC(6),ZC(6)
231   HC(1)= 7:VC(1)= 4:ZC(1)= 4
232   HC(2)=10:VC(2)= 6:ZC(2)=25
233   HC(3)= 5:VC(3)= 7:ZC(3)=10
234   HC(4)= 6:VC(4)= 8:ZC(4)= 5
235   HC(5)= 7:VC(5)= 9:ZC(5)= 5
236   HC(6)=29:VC(6)=11:ZC(6)= 1
400 ' -----------
402 ' Mostra Tela
404 ' -----------
406 CR = 1
408 SEGUE$ = "NAO"
410 CLS
412 PRINT TT$
414 PRINT"------> CADASTRAMENTO MATERIAIS <-------"
416 LOCATE 0,4:PRINT"Codigo:"
418 LOCATE 0,6:PRINT"Descricao:"
420 LOCATE 0,7:PRINT"Tipo:"
422 LOCATE 0,8:PRINT"Saldo:"
424 LOCATE 0,9:PRINT"Minimo:"
426 PRINT
428 PRINT "Posso Gravar no Disco? (S/N): :"
500 ' -------------
502 ' Mostra Valores
504 ' -------------
516 LOCATE  7,4:PRINT EC$":"
518 LOCATE 10,6:PRINT ED$":"
520 LOCATE  5,7:PRINT ET$":"
522 LOCATE  6,8:PRINT ES$":"
524 LOCATE  7,9:PRINT EM$":"
600 ' --------------------
602 ' Entrada de Dados - 1
604 ' --------------------
```

```
610 LOCATE HC(CR),VC(CR):CS=ZC(CR):GOSUB 1000
620   IF ASC(A$) = 27 THEN 950
622   IF ASC(A$) = 13 THEN 710
624   IF ASC(A$) = 28 THEN 710
626   IF ASC(A$) = 31 THEN 710
628   IF ASC(A$) = 29 THEN 720
630   IF ASC(A$) = 30 THEN 720
650 GOTO 730
710 ' ------> Avanca o Cursor
712     CR = CR + 1
713     IF SEGUE$ = "NAO" THEN CR = 1
714     IF CR > 6 THEN CR = 6
716     GOTO 600
720 ' ------> Retrocede o Cursor
722     CR = CR - 1
723     IF SEGUE$ = "NAO" THEN CR = 1
724     IF CR < 1 THEN CR = 1
726     GOTO 600
728 '
730 ON CR GOSUB 810,820,830,840,850,860
732  CR = CR + 1
733   IF SEGUE$ = "NAO" THEN CR = 1
734    GOTO 500
810 ' ------> CR=1 CODIGO DO MATERIAL
811     GOSUB 1200
812      IF C% = 0 THEN EC$ = A$:REG%=FIRST%:
                                 SEGUE$="SIM":RETURN
814       LOCATE 2,5:PRINT "*** Material ja
                              Cadastrado ***"
815        GOSUB 5010
816         LOCATE 0,19:GOSUB 5200:LOCATE 2,5:
                                   PRINT SPACE$(30)
817          SEGUE$ = "NAO"
818           RETURN
820 ' ------> CR=2 DESCRICAO DO MATERIAL
822            ED$ = A$
824            RETURN
830 ' ------> CR=3 TIPO DO MATERIAL
832            ET$ = A$
834            RETURN
840 ' ------> CR=4 SALDO EM ESTOQUE
842            ES$ = STR$(VAL(A$))
844            RETURN
850 ' ------> CR=5 ESTOQUE MINIMO
852            EM$ = STR$(VAL(A$))
854            RETURN
860 ' ------> CR=6 RESPOSTA
862      IF A$ = "S" OR A$ = "s" THEN GOTO 868
```

```
864        CR = 0
866        SEGUE$ = "NAO"
868        IF SEGUE$ = "NAO" THEN RETURN
870 GOSUB 1300
872 LSET CD$ = EC$
873 LSET DC$ = ED$
874 LSET TC$ = ET$
875 LSET SR$ = MKS$(VAL(ES$))
876 LSET SL$ = MKS$(0)
877 LSET QM$ = MKS$(VAL(EM$))
878 LSET UE$ = SPACE$(6)
879 LSET US$ = SPACE$(6)
880  PUT #NARQ,REG%
881   MX% = MX% + 1
882    GET #1,1
884     LSET RC$ = MKI$(MX%)
886      PUT #1,1
888       SEGUE$ = "NAO"
890        RETURN
950 CLOSE
952 PP$ = "COMAT.BAS"
954 GOSUB 5100

MERGE "ROTENT.FNT"
MERGE "ROTPROC.FNT"
MERGE "ROINS.FNT"
MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "QQUER.FNT"
MERGE "MAPAACC.FNT"
MERGE "ABREARQ.FNT"
DELETE 6030-6036
SAVE "CMCAD.BAS"
```

Programa CMMOV.BAS

```
100 ' ==================================
102 ' CMMOV.BAS = Movimento de Materiais
104 ' ==================================
106 ' R.M.Watanabe                06/04/87
200 ' ------------------
202 ' Parametros Iniciais
204 ' ------------------
206   DR$ = "A:"
208   NARQ = 1
210   GOSUB 6010:GET #1,1
400 ' ------------------
402 ' Formatacao da tela
404 ' ------------------
```

```
406 CLS
408 PRINT TT$
410 PRINT"-------> MOVIMENTO DE MATERIAIS <-------"
412 PRINT
414 PRINT "Escolher uma das opcoes:"
416 PRINT
418 PRINT "1 - Registrar uma Entrada"
420 PRINT "2 - Registrar uma Saida"
422 PRINT "3 - Alterar os Dados de uma Entrada"
424 PRINT "4 - Alterar os Dados de uma Saida"
426 PRINT "5 - Cancelar uma Entrada"
428 PRINT "6 - Encerrar o Processamento"
430 PRINT "7 - Voltar a Tela Anterior"
432 PRINT
434 PRINT "*** Opcao escolhida = ";CHR$(255);
500 ' ----------------
501 ' Entrada da Opcao
502 ' ----------------
510   K$=INKEY$:IF K$="" THEN GOTO 510
520    IF K$<"1" OR K$>"7" THEN GOTO 510
530     PRINT
540      PRINT "A opcao escolhida foi:"K$
560       OPCAO = VAL(K$)
690 ON OPCAO GOTO 710,720,730,740,750,760,770
710 ' ----------------------------
712 ' OPCAO = 1  Registrar Entrada
714 ' ----------------------------
716   PP$="CMREGENT.BAS":GOSUB 5100
720 ' --------------------------
722 ' OPCAO = 2  Registrar Saida
724 ' --------------------------
726   PP$="CMREGSAI.BAS":GOSUB 5100
730 ' --------------------------
732 ' OPCAO = 3  Alterar Entrada
734 ' --------------------------
736   PP$="CMALTENT.BAS":GOSUB 5100
740 ' ------------------------
742 ' OPCAO = 4  Alterar Saida
744 ' ------------------------
746   PP$="CMALTSAI.BAS":GOSUB 5100
750 ' ---------------------------
752 ' OPCAO = 5  Cancelar Entrada
754 ' ---------------------------
756   PP$="CMCANENT.BAS":GOSUB 5100
760 ' -------------------------
762 ' OPCAO = 6  Cancelar Saida
764 ' -------------------------
766   PP$="CMCANSAI.BAS":GOSUB 5100
```

```
770 ' -----------------
772 ' OPCAO = 7  Voltar
774 ' -----------------
776   PP$="COMAT.BAS":GOSUB 5100
```

```
MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "ABREARQ.FNT"
DELETE 6020-6036
SAVE "CMMOV.BAS"
```

Programa CMREGENT.BAS

```
100 ' ==================================
102 ' CMREGENT.BAS = Registra Entradas
104 ' ==================================
106 ' Este programa registra as Entradas
108 ' no Almoxarifado.
112 '
114 ' R.M.Watanabe 07/04/87
116 '
120 ' Este Programa e identico ao
122 ' CMREGSAI.BAS com excessao das
124 ' linhas 414 e 894.
200 ' -------------------
201 ' Parametros Iniciais
202 ' -------------------
203   MAXFILES = 3
204   DR$ = "A:"
206   NARQ = 1
208   GOSUB 6010:GET #NARQ,1
210   MX% = CVI(RC$)
211   MV% = CVI(RM$)
212   NARQ = 2:GOSUB 6020
214   NARQ = 3:GOSUB 6030
215 NARQ = 2
216 ED$ = SPACE$(2)
217 EM$ = SPACE$(2)
218 EA$ = SPACE$(2)
219 EC$ = SPACE$(4)
220 ET$ = SPACE$(25)
221 ES$ = SPACE$(5)
222 EE$ = SPACE$(5)
223 EH$ = SPACE$(10)
230 DIM HC(7),VC(7),ZC(7)
231   HC(1)=16:VC(1)= 4:ZC(1)= 2
232   HC(2)=19:VC(2)= 4:ZC(2)= 2
233   HC(3)=22:VC(3)= 4:ZC(3)= 2
```

```
234   HC(4)=19:VC(4)= 5:ZC(4)= 4
235   HC(5)=22:VC(5)=11:ZC(5)= 5
236   HC(6)=10:VC(6)=12:ZC(6)=10
238   HC(7)=29:VC(7)=14:ZC(7)= 1
400 ' -----------
402 ' Mostra Tela
404 ' -----------
406   CR = 1
408   SEGUE$ = "NAO"
410 CLS
412 PRINT TT$
414 PRINT"------> REGISTRO DE ENTRADAS <----------"
416 LOCATE 0,4:PRINT"Data da Entrada:  /  /  :"
417 LOCATE 0,5:PRINT"Codigo do Material:     :"
418 LOCATE 0,8:PRINT"Descricao:"SPACE$(25)":"
419 LOCATE 0,9:PRINT"Saldo no Estoque:       :"
420 LOCATE 0,11:PRINT"Quantidade da Entrada:      :"
421 LOCATE 0,12:PRINT"Historico:"SPACE$(10)":"
426 PRINT
428 PRINT "Posso Gravar no Disco? (S/N): :"
500 ' --------------
502 ' Mostra Valores
504 ' --------------
516 LOCATE 16, 4:PRINT ED$
517 LOCATE 19, 4:PRINT EM$
518 LOCATE 22, 4:PRINT EA$
519 LOCATE 19, 5:PRINT EC$
520 LOCATE 10, 8:PRINT ET$
521 LOCATE 17, 9:PRINT ES$
522 LOCATE 22,11:PRINT EE$
524 LOCATE 10,12:PRINT EH$
600 ' --------------------
602 ' Entrada de Dados - 1
604 ' --------------------
610 LOCATE HC(CR),VC(CR):CS=ZC(CR):GOSUB 1000
620 IF ASC(A$) = 27 THEN 950
622 IF ASC(A$) = 13 THEN 710
624 IF ASC(A$) = 28 THEN 710
626 IF ASC(A$) = 31 THEN 710
628 IF ASC(A$) = 29 THEN 720
630 IF ASC(A$) = 30 THEN 720
650 GOTO 730
710 ' ------> Avanca o Cursor
712     CR = CR + 1
714     IF CR > 7 THEN CR = 7
716     GOTO 600
720 ' ------> Retrocede o Cursor
722     CR = CR - 1
```

```
724       IF CR < 1 THEN CR = 1
726       GOTO 600
728 '
730 ON CR GOSUB 810,820,830,840,850,860,870
732 CR = CR + 1
734 GOTO 500
810 ' ------> CR=1 Dia
812       IF VAL(A$)<1 OR VAL(A$)>31 THEN GOSUB 5300:
             CR=0:SEGUE$="NAO":RETURN
814       ED$ = A$
816       RETURN
820 ' ------> CR=2 Mes
822       IF VAL(A$)<1 OR VAL(A$)>12 THEN GOSUB 5300:
             CR=1:SEGUE$="NAO":RETURN
824       EM$ = A$
826       RETURN
830 ' ------> CR=3 Ano
834       EA$ = A$
836       RETURN
840 ' ------> CR=4 Codigo do Material
841       GOSUB 1200
842        IF C% <> 0 THEN EC$=A$:ET$=DC$:
                        ES$=STR$(CVS(SR$)+CVS(SL$)):
                        SEGUE$="SIM":REG%=C%:RETURN
843       LOCATE 7,6:PRINT "*** Material nao
                              Cadastrado ***"
844        GOSUB 5010
845         LOCATE 0,19:GOSUB 5200:LOCATE 7,6:
                                    PRINT SPACE$(30)
846          SEGUE$ = "NAO"
847           CR = 3
848            RETURN
850 ' ------> CR=5 Quantidade da Entrada
852       IF VAL(A$) = 0 THEN GOSUB 5300:CR=1:RETURN
854        EE$ = A$
856         RETURN
860 ' ------> CR=6 Historico
864       EH$ = A$
865        RETURN
870 ' ------> Resposta
872       IF A$ = "S" OR A$ = "s" THEN GOTO 878
874        CR = 0
876         SEGUE$ = "NAO"
878          IF SEGUE$ = "NAO" THEN RETURN
880 ' ------> Analisa se os dados sao validos
882 A$ = ED$:GOSUB 810
884 A$ = EM$:GOSUB 820
887 A$ = EE$:GOSUB 850
```

```
888 IF SEGUE$ = "NAO" THEN RETURN
890     LSET MD$ = ED$+EM$+EA$
891     LSET MH$ = EH$
892     LSET MC$ = EC$
893     LSET MQ$ = MKS$(VAL(EE$))
894     LSET MV$ = "E"
895 MV% = MV% + 1
896 PUT #3,MV%
897     GET #1,1
898     LSET RM$ = MKI$(MV%)
899     PUT #1,1
900
901     GET #2,REG%
902     LSET UE$ = UD$ + UM$ + UA$
903     PUT #2,REG%
910 SEGUE$ = "NAO"
911 CR = 0
912 RETURN
950 ' ------> Saida do Programa
952             CLOSE
954             PP$ = "COMAT.BAS"
956             GOTO 5100

MERGE "ROTENT.FNT"
MERGE "ROTPROC.FNT"
MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "QQUER.FNT"
MERGE "ERRO.FNT"
MERGE "ABREARQ.FNT"
SAVE "CMREGENT.BAS"
```

Programa CMREGSAI.BAS

Desenvolver este programa como exercício.

Ele é semelhante ao programa CMREGENT.BAS, com exceção das linhas 414, 894 e 902.

Programa CMALTENT.BAS

```
100 ' ================================
102 ' CMALTENT.BAS = Altera Entrada
104 ' ================================
106 ' Este programa permite alterar
108 ' os dados de uma entrada.
112 '
114 ' R.M.Watanabe 07/04/87
116 '
```

```
120 ' Este programa e identico ao
122 ' CMALTSAI.BAS com excecao das
124 ' linhas 203 e 414.
200 ' -------------------
201 ' Parametros Iniciais
202 ' -------------------
203   MAXFILES = 3
204   TIPO$ = "E"
205   DR$ = "A:"
206   NARQ = 1
208     GOSUB 6010:GET #NARQ,1
210     MX% = CVI(RC$)
211     MV% = CVI(RM$)
212   NARQ = 2:GOSUB 6020
214   NARQ = 3:GOSUB 6030
215 NARQ = 2
216 ED$ = SPACE$(2)
217 EM$ = SPACE$(2)
218 EA$ = SPACE$(2)
219 EC$ = SPACE$(4)
220 ET$ = SPACE$(25)
221 ES$ = SPACE$(5)
222 EE$ = SPACE$(5)
223 EH$ = SPACE$(10)
230 DIM HC(7),VC(7),ZC(7)
231   HC(1)=16:VC(1)= 4:ZC(1)= 2
232   HC(2)=19:VC(2)= 4:ZC(2)= 2
233   HC(3)=22:VC(3)= 4:ZC(3)= 2
234   HC(4)=19:VC(4)= 5:ZC(4)= 4
235   HC(5)=22:VC(5)=11:ZC(5)= 5
236   HC(6)=10:VC(6)=12:ZC(6)=10
238   HC(7)=29:VC(7)=14:ZC(7)= 1
400 ' -----------
402 ' Mostra Tela
404 ' -----------
406   CR = 1
408   SEGUE$ = "NAO"
410 CLS
412 PRINT TT$
414 PRINT"------> ALTERACAO DE ENTRADA <----------"
416 LOCATE 0,4:PRINT"Data da Entrada:  /  /  :"
417 LOCATE 0,5:PRINT"Codigo do Material:     :"
418 LOCATE 0,8:PRINT"Descricao:"SPACE$(25)":"
419 LOCATE 0,9:PRINT"Saldo no Estoque:       :"
420 LOCATE 0,11:PRINT"Quantidade da Entrada:      :"
421 LOCATE 0,12:PRINT"Historico:"SPACE$(10)":"
426 PRINT
428 PRINT "Posso Gravar no Disco? (S/N): :"
```

```
500 ' --------------
502 ' Mostra Valores
504 ' --------------
516 LOCATE 16, 4:PRINT ED$
517 LOCATE 19, 4:PRINT EM$
518 LOCATE 22, 4:PRINT EA$
519 LOCATE 19, 5:PRINT EC$
520 LOCATE 10, 8:PRINT ET$
521 LOCATE 17, 9:PRINT ES$
522 LOCATE 22,11:PRINT EE$
524 LOCATE 10,12:PRINT EH$
600 ' --------------------
602 ' Entrada de Dados - 1
604 ' --------------------
610 LOCATE HC(CR),VC(CR):CS=ZC(CR):GOSUB 1000
612 IF CR = 1 AND ASC(A$) = 30 THEN REG% = REG% - 1:
                                    GOTO 750
614 IF CR = 1 AND ASC(A$) = 31 THEN REG% = REG% + 1:
                                    GOTO 750
620 IF ASC(A$) = 27 THEN 950
622 IF ASC(A$) = 13 THEN 710
624 IF ASC(A$) = 28 THEN 710
626 IF ASC(A$) = 31 THEN 710
628 IF ASC(A$) = 29 THEN 720
630 IF ASC(A$) = 30 THEN 720
650 GOTO 730
710 ' ------> Avanca o Cursor
712             CR = CR + 1
714             IF CR > 7 THEN CR = 7
716             GOTO 600
720 ' ------> Retrocede o Cursor
722             CR = CR - 1
724             IF CR < 1 THEN CR = 1
726             GOTO 600
728 '
730 ON CR GOSUB 810,820,830,840,850,860,870
732    CR = CR + 1
734    GOTO 500
736 '
750 IF REG% < 1 OR REG% > MV% THEN GOTO 600
751  GET #3,REG%
752   IF TIPO$ <> MV$ THEN GOTO 612
753    ED$=LEFT$(MD$,2):EM$=MID$(MD$,3,2):
                         EA$=RIGHT$(MD$,2)
754     EC$ = MC$
755      EE$ = STR$(CVS(MQ$))
756       EH$ = MH$
757        A$ = EC$
```

```
758         GOSUB 840
759          GOTO 400
810 ' ------> CR=1 Dia
812      IF VAL(A$)<1 OR VAL(A$)>31 THEN GOSUB 5300:
            CR=0:SEGUE$="NAO":RETURN
814      ED$ = A$
816      RETURN
820 ' ------> CR=2 Mes
822      IF VAL(A$)<1 OR VAL(A$)>12 THEN GOSUB 5300:
            CR=1:SEGUE$="NAO":RETURN
824      EM$ = A$
826      RETURN
830 ' ------> CR=3 Ano
831      EA$=A$:A$=ED$+EM$+EA$:NARQ=3:GOSUB 1400:NARQ=2
832      IF REG%=0 THEN LOCATE 0,6:PRINT "** Nao houve
                        Movimento nesta data**":
                        GOSUB 5010:LOCATE 0,19:G
834      EC$ = MC$
836      EE$ = STR$(CVS(MQ$))
838      EH$ = MH$
839      A$ = MC$
840 ' ------> CR=4 Codigo do Material
841      GOSUB 1200
842      IF C% <> 0 THEN EC$=A$:ET$=DC$:
                         ES$=STR$(CVS(SR$)+CVS(SL$)):
                         SEGUE$="SIM":RETURN
843      LOCATE 7,6:PRINT "*** Material nao
                              Cadastrado ***"
844      GOSUB 5010
845      LOCATE 0,19:GOSUB 5200:LOCATE 7,6:
         PRINT SPACE$(30)
846      SEGUE$ = "NAO"
847      CR = 3
848      RETURN
850 ' ------> CR=5 Quantidade da Entrada
852      IF VAL(A$) = 0 THEN GOSUB 5300:CR=1:RETURN
854      EE$ = A$
856      RETURN
860 ' ------> CR=6 Historico
864      EH$ = A$
865      RETURN
870 ' ------> Resposta
872      IF A$ = "S" OR A$ = "s" THEN GOTO 878
874      CR = 0
876      SEGUE$ = "NAO"
878      IF SEGUE$ = "NAO" THEN RETURN
880 ' ------> Analisa se os dados sao validos
882   A$ = ED$:GOSUB 810
```

```
884   A$ = EM$:GOSUB 820
887   A$ = EE$:GOSUB 850
888      IF SEGUE$ = "NAO" THEN RETURN
891         LSET MD$ = ED$+EM$+EA$
892         LSET MH$ = EH$
893         LSET MC$ = EC$
894         LSET MQ$ = MKS$(VAL(EE$))
896         PUT #3,REG%
900         SEGUE$ = "NAO"
901         CR = 0
902         RETURN
950 CLOSE
952  PP$ = "COMAT.BAS"
954   GOSUB 5100

MERGE "ROTENT.FNT"
MERGE "ROTPROC.FNT"
MERGE "ROTPMOV.FNT"
MERGE "SONS.FNT"
MERGE "ESPERE.FNT"
MERGE "OQUER.FNT"
MERGE "ERRO.FNT"
MERGE "ABREARQ.FNT"
SAVE "CMALTENT.BAS"
```

Programa CMALTSAI.BAS

Desenvolver este programa como exercício.

Ele é semelhante ao CMALTENT.BAS, com exceção das linhas 203 e 414.

Programa CMCANENT.BAS

Desenvolver este programa como exercício.

Programa CMCANSAI.BAS

Desenvolver este programa como exercício.

Programa CMREL.BAS

Desenvolver este programa como exercício.

Ele é parecido com o COMAT.BAS .

Programa CMREL1.BAS

```
100 ' ========================================
102 ' CMREL1.BAS = Emite Relacao de Materiais
104 ' ========================================
106 ' Este programa emite o relatorio
108 ' RELACAO DE MATERIAIS
110 '
112 ' R.M.Watanabe                    20/04/87
114 '
200 ' --------------------
202 ' Parametros Iniciais
204 ' --------------------
206   DR$ = "A:"
208   NARQ = 1:GOSUB 6010:GET #1,1
210   MX% = CVI(RC$)
214   NARQ = 2:GOSUB 6020
216     JIC$ = "NAO"
218     PAGINA = 1
220     MLINHA = 54
222     FM$ = "#####"
400 ' -----------
402 ' Mostra Tela
404 ' -----------
410   CLS
412    PRINT TT$
414     PRINT "------> IMPRESSAO DO REL 1 <-----------"
416     GOSUB 5350
600 ' ---------------------------------
602 ' Nucleo de Emissao do Relatorio
604 ' ---------------------------------
610   FOR REG% = 1 TO MX%
614      GET 2,REG%
620      IF JIC$ = "NAO" THEN GOSUB 920
630      LPRINT SPC(1)CD$SPC(2)DC$SPC(1);TC$
640      IF NLINHA > MLINHA THEN GOSUB 910
642      LOCATE 5+30*REG%/MX%,18:PRINT ">"
650      NEXT REG%
660   CLOSE
672   LPRINT STRING$(43,"=")
800   PP$ = "CMREL.BAS":GOSUB 5100
910 ' --------------
911 ' Salto de Pagina
912 ' --------------
913   LPRINT STRING$(43,"-")
914   LPRINT CHR$(12);
915   JIC$ = "NAO"
916   NLINHA = 0
```

```
917   PAGINA = PAGINA + 1
919   RETURN
920 ' -------------------
921 ' Imprime o Cabecalho
922 ' -------------------
924 LPRINT "Metalurgina METAL Ltda."
925 LPRINT
926 LPRINT "Controle de Materiais - MATERIAIS"TAB(37)
        "FL-";
927 LPRINT USING "###";PAGINA
928 LPRINT
929 LPRINT "Data da Emissao: "UD$"/"UM$"/"UA$
930 LPRINT STRING$(43,"-")
931 LPRINT "CODIGO DESCRICAO"SPC(17)"TIPO"
932 LPRINT
933 JIC$ = "SIM"
934 RETURN

MERGE "ESPERE.FNT"
MERGE "MAPAACC.FNT"
MERGE "ABREARQ.FNT"
DELETE 6030-6036
SAVE "CMREL1.BAS"
```

APÊNDICE I - CONVERSÃO DE COMANDOS

A linguagem BASIC, embora universal, pode apresentar algumas diferenças em função do equipamento onde esteja implementada.

Em particular, os Basics dos equipamentos compatíveis com o Apple-CP/M, MSX e IBM-PC apresentam pequenas diferenças. Os exercícios e exemplos do livro foram elaborados no BASIC MSX, de modo que os leitores que possuem outros tipos de microcomputadores devem estudar as diferenças cuidadosamente antes de se lançar a elaboração do Sistema de Controle de Estoque. As diferenças principais são as seguintes:

1 - As teclas de movimentação do cursor possuem códigos ASCII diferentes. Veja a tabela seguinte:

TECLA	MSX	IBM-PC	APPLE-CP/M
→	28	0 77	21
↓	31	0 80	não tem
←	29	0 75	8
↑	30	0 72	não tem
HOME	11	0 71	não tem

Os exemplos do livro devem ser adaptados.

Para o IBM-PC, a função INKEY$ retornará uma string de 2 caracteres, com o primeiro igual a 0. O programa poderá verificar o segundo com:

624 IF ASC(RIGHT$(A$,1))=77 THEN GOTO 710

em vez de:

624 IF ASC(A$)=28 THEN GOTO 710

Para o Apple-CP/M, que não dispõe de determinadas teclas, há a opção de escolher alguma outra tecla para simular as inexistentes.

2 - O comando OPEN possue 2 sintaxes básicas:

I - OPEN "nome" FOR modo AS #arquivo LEN=tamanho

II - OPEN "modo",#arquivo,"nome", tamanho

Para o MSX, vale a sintaxe I e o nome do arquivo pode conter o do dispositivo a ser usado:

CAS:	para a fita cassete
CRT:	para o monitor de vídeo
LPT:	para a impressora
GRP:	para o monitor de vídeo no modo gráfico
A:, B: C:, D: E:, F:	para os drives de discos

No MSX os arquivos sequenciais são definidos com:

OPEN "nome" FOR modo as #arquivo

O modo pode ser INPUT, para ser lido, OUTPUT, para ser escrito ou APPEND, para ser acrescido de dados. Por exemplo:

```
100 OPEN "A:CADASTRO.ARQ" FOR INPUT AS #1
```

Os arquivos randômicos são definidos com:

OPEN "nome" as #arquivo LEN=tamanho

Por exemplo:

```
200 OPEN "A:CADASTRO.ARQ" AS #1 LEN=78
```

No Apple-CP/M os arquivos sequenciais são definidos com:

OPEN "modo", #arquivo,"nome"

O modo pode ser I, para ser lido, ou O para ser escrito. Por exemplo:

```
100 OPEN "I",#1,"A:CADASTRO_ARQ"
```

Os arquivos randômicos são definidos com:

OPEN "R", #arquivo,"nome", tamanho

Por exemplo:

```
200 PEN "R",#1,"A:CADASTRO.ARQ",78
```

No IBM-PC podem ser empregadas tanto uma sintaxe como a outra, indistintamente. Os dispositivos, além dos disk drives, são os seguintes:

CAS1:	para a fita cassete
COM1:	para 1ª porta de comunicação
COM2:	para 2ª porta de comunicação
KYBD:	para o teclado
LPT1:	para a 1ª impressora
LPT2:	para a 2ª impressora
LPT3:	para a 3ª impressora
SCRN:	para o monitor do vídeo

3 - O tamanho máximo do nome da variável depende do sistema.

No MSX, por exemplo, os nomes SALDO e SALARIO embora pareçam se referir à variáveis distintas, referem-se à variável SA, pois apenas as 2 primeiras letras dos nomes das variáveis são consideradas.

Em resumo:

	MSX	IBM-PC	APPLE-CP/M
Pode ter até :	254	254	254
Considera apenas:	2	40	40

4 - Os comandos de posicionamento do cursor também são diferentes.

No MSX, o comando é LOCATE e sua sintaxe é a seguinte:

LOCATE coluna,linha,cursor

No IBM-PC, o comando também é o LOCATE porém a sua sintaxe é diferente e mais completa:

LOCATE coluna,linha,cursor,início,fim

Os parâmetros início e fim indicam o tamanho do cursor.

No Apple-CP/M, os comandos são o VTAB e o HTAB com as seguintes sintaxes:

VTAB linha
HTAB coluna

Os valores possíveis disponíveis na configuração mínima das máquinas estão resumidos a seguir.

	MSX	IBM-PC	APPLE-CP/M
Coluna	0 a 39	1 a 80	1 a 40
Linha	0 a 23	1 a 24	1 a 24
Cursor	0 ou 1	0 ou 1	não tem

5 - A especificação da quantidade máxima de arquivos simultâneamente abertos se faz de forma diferente.

No MSX, utiliza-se o comando MAXFILES. Por exemplo:

```
100 MAXFILES = 5
```

No IBM-PC e no Apple-CP/M se faz durante a carga do BASIC na memória com:

BASICA /F:5

ou

MBASIC /F:5

Em todos os casos, a quantidade de arquivos pode ser entre 1 e 15, sendo o "default" igual a 3.

A tabela a seguir apresenta a relação de todos os comandos e funções dos Basics:

TABELA COMPARATIVA ENTRE BASICS

IBM-PC	MSX	APPLE-CP/M	PISTA
ABS(r)	=	=	
ASC(c$)	=	=	
ATN(r)	=	=	
AUTO l,i	=	=	
não tem	BASE(i)	não tem	
BEEP	=	=	
não tem	não tem	BEEP i,t	
não tem	BIN$(i)	não tem	
BLOAD	=	não tem	
BSAVE	=	não tem	
não tem	não tem	BUTTON(p)	
CALL	=	=	
não tem	não tem	CALL %	
CDBL(d$)	=	=	
CHAIN	não tem	=	

CHR$(a)	=	=	
CINT(r)	=	=	
CIRCLE	=	não tem	
não tem	CIRCLE STEP	não tem	
não tem	não tem	CLEAR e,i	
CLEAR i,i	não tem	não tem	
não tem	CLEAR e,i	não tem	
não tem	CLIAD	não tem	
CLOSE	=	=	
CLS	=	não tem	HOME
COLOR 1,f,b	=	não tem	
COLOR f,p	não tem	não tem	
não tem	não tem	COLOR=i	
COM(p)	não tem	não tem	
COMMON	não tem	=	
CONT	=	=	
COS(a)	=	=	
não tem	CSAVE	não tem	
CSNG(d)	=	=	
CSRLIN 1 a 25	= 0 a 24	não tem	VPOS
CVD(d$)	=	=	
CVI(i$)	=	=	
CVS(s$)	=	=	
DATA	=	=	
DATE$	não tem	não tem	
DATE$	não tem	não tem	
DEF FN	=	=	
DEF SEG	não tem	não tem	
DEF USR	=	=	
DEFDBL	=	=	
DEFINT	=	=	
DEFSGN	=	=	
DEFSTR	=	=	
DELETE i-i	=	não tem	DEL
não tem	não tem	DEL i,i	DELETE
DIM	=	=	
DRAW	=	não tem	
EDIT	não tem	=	
END	=	=	
EOF	=	=	
ERASE	=	=	
ERL	=	=	
ERR	=	=	
ERROR	=	=	
EXP(n)	=	=	
FIX(n)	=	=	
FIELD	=	=	
FILES	=	=	

FOR	=	=	
FRE(i)	=	=	
FRE("")	=	=	
GET #f	=	=	
GET #p	não tem	não tem	
GET m	não tem	não tem	BLOAD
não tem	não tem	GET c$	INKEY$
GOSUB	=	=	
GOTO	=	=	
não tem	não tem	GR x,y	SCREEN
HEX$(i)	=	=	
não tem	não tem	HLIN	LINE
não tem	não tem	HOME	CLS
não tem	não tem	HPLOT	PRESET
não tem	não tem	HTAB	LOCATE
IF	=	=	
INKEY$	=	=	
INP(p)	=	não tem	
INPUT	=	=	
INPUT$f	=	=	
INPUT$	=	=	
INSTR	=	=	
INT(r)	=	=	
não tem	INTERVAL	não tem	
não tem	não tem	INVERSE	COLOR
KEY	=	não tem	
KILL	=	=	
LEFT$	=	=	
LEN(c$)	=	=	
LET	=	=	
LINE	=	não tem	HLIN
não tem	LINE STEP	não tem	
LINE INPUT	=	=	
LINE INPUT #f	=	=	
LIST	=	=	
LIST ,disp	não tem	não tem	LLIST
LLIST	=	=	
LEN(c$)	=	=	
LOAD	=	=	
LOC(f)	=	=	
LOCATE l,c	diferente	não tem	VTAB,HTAB
diferente	LOCATE c,l	não tem	HTAB,VTAB
LOF(f)	=	=	
LOG(r)	=	=	
LPOS(p)	=	=	
LPRINT	=	=	
LPRINT USING	=	=	
LSET	=	=	

não tem	MAXFILES=n	não tem	
MERGE	=	=	
MID$	=	=	
MID$	=	=	
MKD$(d)	=	=	
MKI$(i)	=	=	
MKS$(r)	=	=	
MOTOR (0) : (1)	diferente	não tem	
diferente	MOTOR ON : OFF	não tem	
NAME	=	=	
NEW	=	=	
NEXT	=	=	
NEXT v	=	não tem	
não tem	não tem	NORMAL	SCREEN
não tem	não tem	NOTRACE	TRON
OCT$(i)	=	=	
ON COM(p)	não tem	não tem	
ON ERROR	=	=	
ON x GOSUB	=	=	
ON x GOTO	=	=	
não tem	ON INTERVAL	não tem	
diferente	ON KEY GOSUB	não tem	
ON KEY(n)	diferente	não tem	
ON PEN	não tem	não tem	
não tem	ON SPRITE	não tem	
não tem	ON STOP	não tem	
diferente	ON SRTRING	não tem	
ON STRING(n)	diferente	não tem	
OPEN FOR	=	diferente	
OPEN #f	diferente	=	
OPEN "COM:	não tem	não tem	
OPTION BASE	não tem	=	
OUT p,v	=	não tem	
não tem	PAD(p)	não tem	
PAINT	=	não tem	
não tem	PAINT STEP	não tem	
não tem	PDL(p)	=	
PEEK(i)	=	=	
PEN	não tem	não tem	
PEN	não tem	não tem	
PLAY	não tem	não tem	
não tem	PLAY	não tem	
não tem	não tem	PLOT	
POINT	=	não tem	
POKE	=	=	
não tem	não tem	POP	
POS(x)	=	=	
PRESET	=	não tem	HPLOT

não tem	PRESET STEP	não tem	
PRINT	=	=	
PRINT USING	=	=	
PRINT#f	=	=	
PRINT $f,USING	=	=	
PSET	=	não tem	PLOT
não tem	PSET STEP	não tem	
PUT #f,r	=	=	
PUT m	não tem	não tem	BSAVE
não tem	PUT SPRITE	não tem	
RANDOMIZE	não tem	=	
READ lista	=	=	
REM	=	=	
RENUM	=	=	
RESET	=	=	
RESTORE	=	=	
RESUME	=	=	
RETURN	=	=	
RETURN linha	não tem	não tem	
RIGHT$(c$,a)	=	=	
RND(i)	=	=	
RSET	=	=	
RUN	=	=	
RUN "a",R	=	=	
não tem	RUN i	=	
SAVE "a",A	=	=	
SAVE "a",P	não tem	=	
SCREEN	diferente	não tem	
diferente	SCREEN	não tem	
SCREEN	não tem	não tem	
não tem	não tem	SCRN	POINT
SGN(n)	=	=	
SIN(r)	=	=	
SOUNd f,d	diferente	não tem	BEEP
diferente	SOUND r,e	não tem	BEEP
SPACE$(a)	=	=	
SPC(a)	=	=	
não tem	SPRITE	não tem	
não tem	SPRITE$	não tem	
SQR(n)	=	=	
STICK(x)	=	não tem	PDL
STOP	=	=	
não tem	STOP ON:OFF	não tem	
STR$(a)	=	=	
STRIG ON:OFF	não tem	não tem	
STRIG(p)	=	não tem	
STRIG(p)	=	não tem	
STRING$	=	=	

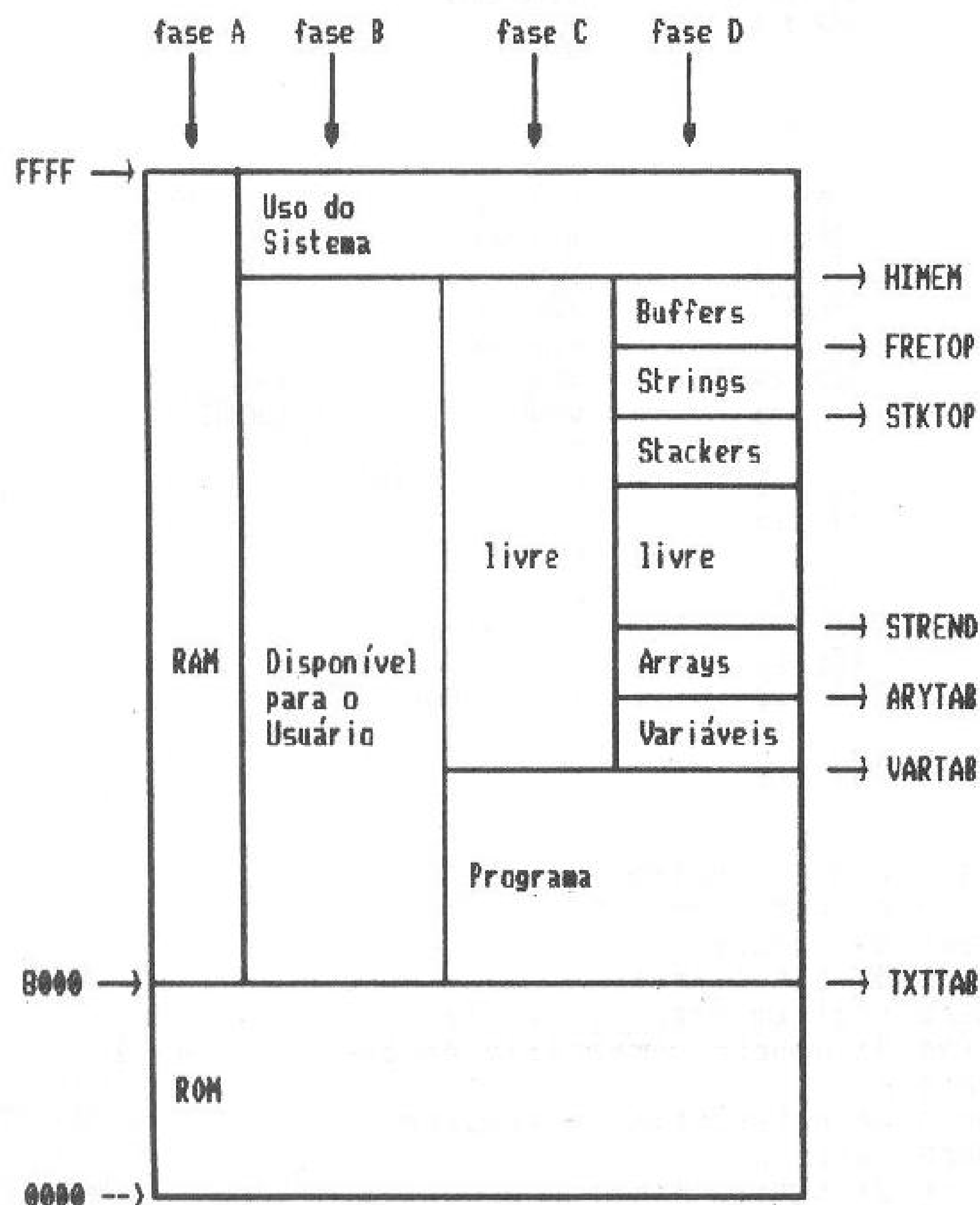
SWAP	=	=	
SYSTEM	não tem	=	CALL
TAB(a)	=	=	
TAN(r)	=	=	
não tem	não tem	TEXT	SCREEN
diferente	TIME	não tem	
diferente	TIME	não tem	
TIME$	diferente	não tem	
TIME$	diferente	não tem	
não tem	não tem	TRACE	TRON
TROFF	=	não tem	NOTRACE
TRON	=	não tem	TRACE
USR(i)	difere te	diferente	
diferente	USR X A	=	
VAL(c$	=	=	
VARPTR(v$)	=	=	
VARPTR(#f)	não tem	=	
VARPTR$($)	não tem	não tem	
não tem	VDP(n)	não tem	
não tem	VDP(n)	não tem	
não tem	não tem	VLIN	LINE
não tem	VPEEK	não tem	
não tem	VPOKE	não tem	
não tem	não tem	VPOS	CSR IN
não tem	não tem	VTAB	LOCATE
WAIT p,n,m	=	=	
WEND	não tem	=	
WHILE	não tem	=	
WIDTH n	=	=	
não tem	não tem	WIDTH l,h	
WIDTH #f,i	não tem	não tem	
WIDTH disp,i	não tem	diferente	
diferente	não tem	WIDTH LPRINT	
WRITE	não tem	=	
WRITE #f	não tem	=	

a = número ASCII (entre 0 e 255)
b = ângulo em radianos
c = número da coluna
c$ = string de caracteres
d = número real em precisão dupla
d$ = string de número compactado de precisão dupla
e = endereço
f = número de referência do arquivo
i = número inteiro
i$ = string de número inteiro
l = número da linha

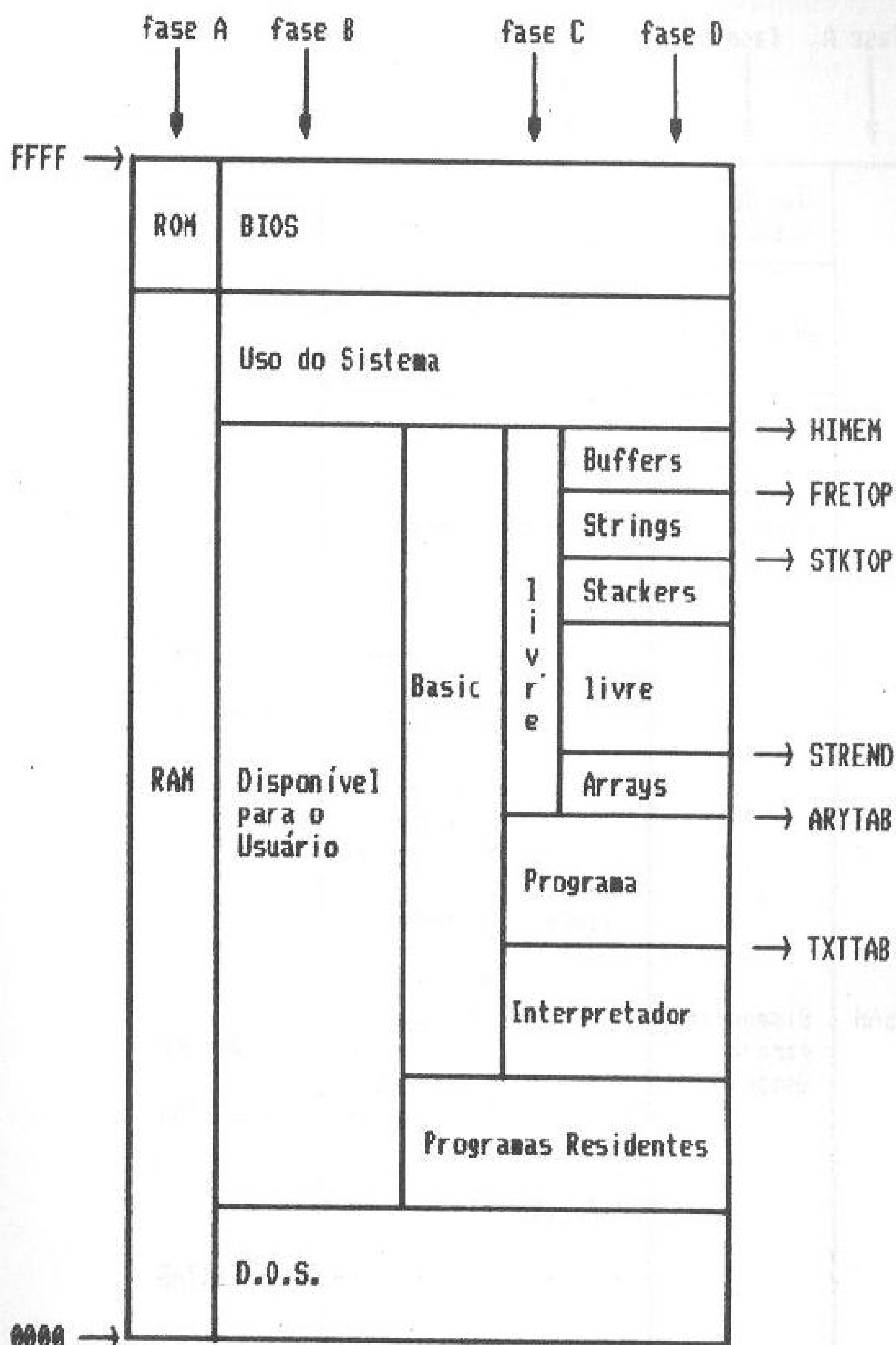
m = matriz
p = número da porta
r = número real
s$ = string de número compactado de precisão simples
t = tempo
v = variável numérica
v$ = nome de uma variável

APÊNDICE II – OCUPAÇÃO DA MEMÓRIA

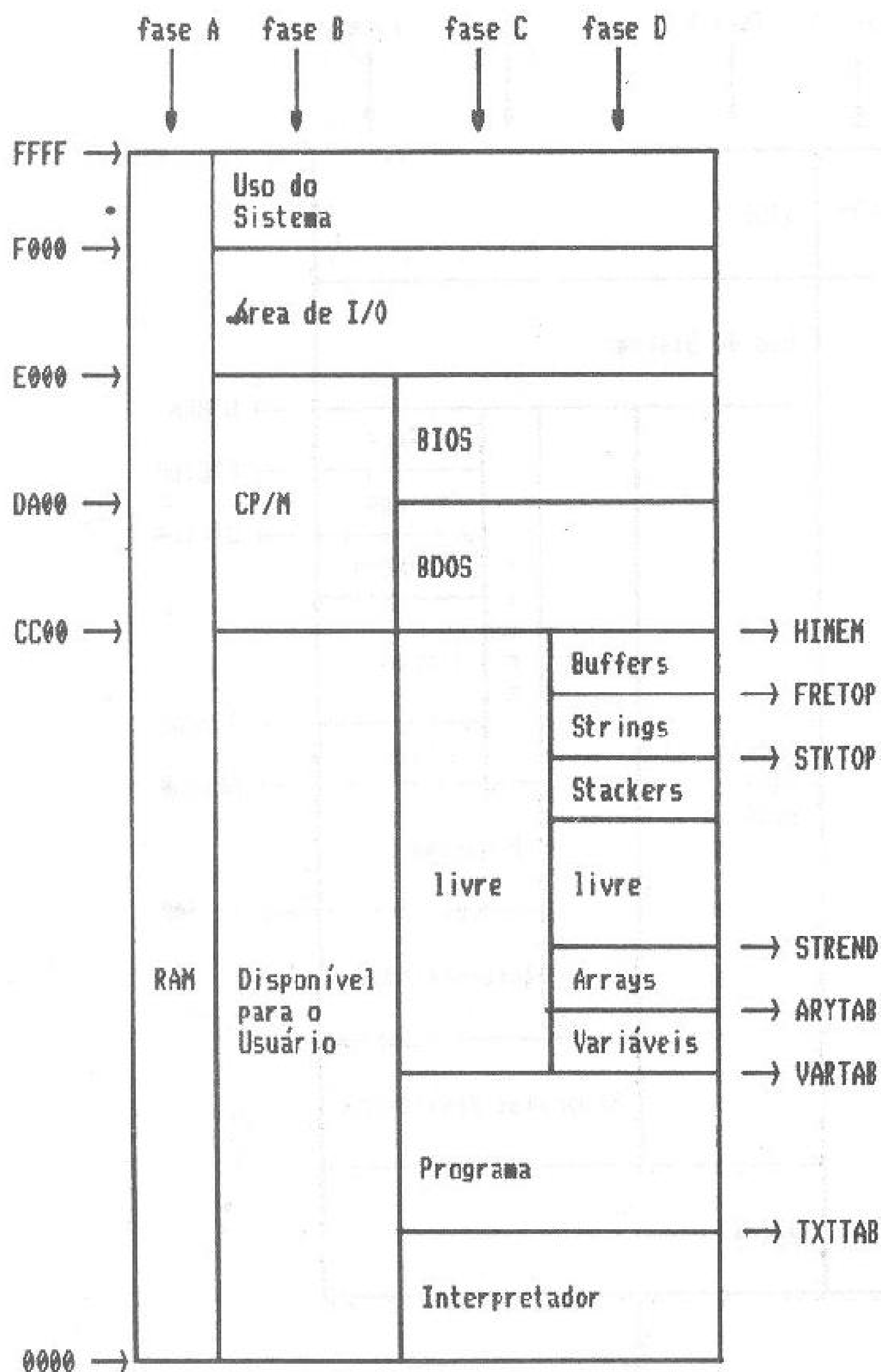
COMPUTADOR MSX:



COMPUTADOR IBM-PC



COMPUTADOR APPLE-CP/M



programação profissional em basic

ROBERTO MASSARU WATANABE, é um dos nomes de maior destaque dentro da informática brasileira. Conhecido já há muito pelos usuários de microcomputadores, seus trabalhos realmente mais notáveis realizaram-se em ambientes de grande porte.

Paralelamente ao seu trabalho como profissional de informática, Watanabe sempre se dedicou a luta pela melhoria da qualidade dos profissionais brasileiros, assumindo tarefas de educador com grande maestria.

Neste livro, sempre preciso e tecnicamente impecável mas nem por isso abandonando a clareza e o didatismo, Watanabe transmite ao leitor em vias de profissionalização uma nova visão da linguagem BASIC. As "boas técnicas" de programação são ensinadas e justificadas, com abordagens sobre análise de sistemas e técnicas de depuração e otimização. Como sub-produto da leitura do livro, o usuário passa a dispor de um programa de Controle de Estoque sobre o qual tem completo domínio, podendo alterá-lo com segurança.

MSX

IBM-PC

MBASIC





EDITORA ALEPH
Caixa Postal 20.707
CEP 01498 - São Paulo-SP.